

ORACLE®

Academy

O que é uma Variável?

- Considere a variável x em uma equação.
- Podemos atribuir qualquer valor a ela.

$$y = -2x + 5$$

$$x = 0$$

$$y = -2 \times 0 + 5$$

$$y = 0 + 5$$

$$y = 5$$

$$x = 2$$

$$y = -2 \times 2 + 5$$

$$y = -4 + 5$$

$$y = 1$$

O que É uma Variável em Java?

Da mesma forma, podemos atribuir valores a variáveis Java.

```
String x = "Alex";  
System.out.println("Meu nome é " + x);
```



"Meu nome é Alex"

Desvantagem de Não Usar Variáveis

- O código não é flexível.
- Para substituir o nome "Alex", você precisa fazer muitas alterações em muitos locais:
 - Edição monótona
 - Risco de deixar passar um "Alex" despercebido

```
System.out.println("Meu nome é Alex");  
System.out.println("Alex é muito bacana");  
System.out.println("Viva o Alex!");  
System.out.println("Aproveite a Avaliação de Alex "  
    + "Dia! Meu nome é Alex. Sei como todos estão  
    interessados em "  
    + "começar a apreciar o Alex no Dia da"  
    + "Avaliação do Alex! Alex, Alex, Alex! Sim "  
    + "Alex!!! Este sou eu! Alex é o melhor encontro de todos  
    os tempos!");
```

Vantagens de Usar Variáveis

- O código torna-se flexível.
 - Lembrar e manipular variáveis
- Para substituir o nome "Alex", você faz uma alteração:
 - Edição eficiente
 - Sem risco de deixar passar um "Alex" despercebido

```
String x = "Sam";
System.out.println("Meu nome é " +x);
System.out.println(x + " é muito bacana!");
System.out.println("Viva " +x + "!");
System.out.println("Aproveite a Avaliação de " +x + " "
    + "Dia! Meu nome é " +x + ". Sei como todos estão
    interessados em "
    + "começar a apreciar " +x + " no Dia da Avaliação de " +x
    + "! " +x + ", " +x + ", " +x + "! Sim "
    + x + "!!! Este sou eu! " +x + " é o melhor pretendente de
    todos os tempos!");
```

Mais Vantagens com Variáveis

Manipule valores muitas vezes de várias maneiras:

- Mude você mesmo os valores diretamente (mostrado a seguir).
- Altere valores calculados de forma programada.
- Mude com base na entrada do usuário.

```
5      String x = "Alex";  
6      x = "Sam";  
7      x = "Nicky";  
8      x = "Encontro Secreto";  
9  
10     "voltar" = x;    //Não é possível fazer isso
```

Muitos Tipos de Variáveis


- As variáveis podem existir para muitos tipos diferentes no Java.
- Aqui estão as variáveis que você viu:

Tipo	Palavra-chave	Exemplo de Valores
Booleano	<code>boolean</code>	<code>true, false</code>
Inteiro	<code>int</code>	<code>1, -10, 20000, 123_456_789</code>
Duplo	<code>double</code>	<code>1.0, -10.0005, 3.141</code>
String	<code>String</code>	<code>"Alex", "Jantei demais."</code>

Declarando uma Variável

- Java é uma “linguagem que envolve muita digitação”.
 - Você precisa **declarar** que tipo de dados sua variável tratará usando **palavras-chave**.

tipo *nome*




```
boolean bool;  
int x;  
duplo y;  
String z;
```

- Depois de você declarar uma variável...
 - Essa variável passa a existir.
 - Não é preciso declará-la novamente.

Opções para Declarar e Atribuir Valores

A. Declare e atribua uma variável em uma única linha.



```
boolean bool = true;
```

B. Declare uma variável em uma linha e atribua um valor mais tarde.

```
boolean bool;  
// ...  
bool = true;
```

Atribuindo Valores Inadequados

Os valores atribuídos devem ser apropriados para o tipo de dados que você declarou.



```
int x = 3;
```



```
int z = "Puppies!";
```

Valores Matemáticos Inapropriados

- Podemos atribuir qualquer valor a x
- Não podemos atribuir um valor String a x
 - Isso não faz sentido!

$$y = -2x + 5$$

$x = \text{"Puppies!"}$

$$y = -2(\text{"Puppies!"}) + 5$$

$$y = ???$$



Erros com Variáveis

- Atribuindo valores inadequados a um tipo de variável

```
int intVar1 = true;
```

- Esquecendo de declarar um tipo de variável

```
intVar3 = 3;
```

- Escrevendo uma variável com a grafia incorreta

```
double doubleVar2;  
doublevAr2 = 2.1;    //O Java faz distinção  
                     entre maiúsculas e  
                     minúsculas
```

Erros com Variáveis

- Declarar a mesma variável duas vezes

```
double doubleVar3;  
double doubleVar3 = 3.1;
```

- Esquecer de atribuir um valor antes de usar uma variável

```
double doubleVar4;  
System.out.println(doubleVar4) ;
```

A atribuição de um valor inicial à variável denomina-se “inicialização”.

Você Pode Ter Notado...

- É possível declarar muitas variáveis em uma única linha.

```
double doubleVar1, doubleVar2, doubleVar3;
```

- É possível atribuir valores ao declarar muitas variáveis.

```
double doubleVar1, doubleVar2, doubleVar3 = 3.1;
```

- É uma questão de preferência pessoal...
 - Declarar toda variável em linhas separadas
 - Declarar todas as variáveis de um tipo específico em uma única linha

Atribuição de Nomes Inadequados a Variáveis



- Você pode atribuir *praticamente* qualquer nome a uma variável.

```
int dsfdsfspoop = 20;    //Ha ha!
```

- Isso seria divertido, mas...
- Será que, ao ler o código, você ou um amigo entenderia quais dados dsfdsfspoop representa?

- Em geral, não é recomendável o uso de nomes muito pequenos.

```
int x = 20;
```

- Isso é muito útil para teste...
- E é uma prática comum em loops pequenos (assunto abordado mais adiante), mas...
- Será que, ao ler o código, você ou um amigo entenderia quais dados x representa?

Atribuição de Nomes Muito Inadequados Variáveis

- As variáveis podem compartilhar o mesmo nome.

```
int x = 20;  
double x = 22.0;  
System.out.println(x); //Que x?
```

- As variáveis não podem começar com números.

```
boolean 1337Hacker = true;
```

- As palavras-chave não podem ser usadas para nomes de variáveis.

```
int continue = 20;
```

- As palavras-chave ficam azuis no NetBeans
- As palavras-chave têm significados especiais no Java.

Convenções de Nomenclatura de Variáveis



- Comece cada variável com uma letra minúscula.
As palavras subsequentes devem ser capitalizadas:
 - `myVariable`
- Escolha nomes que sejam mnemônicos e que indiquem a intenção da variável para o observador casual.
- Lembre-se de que...
 - Os nomes fazem distinção entre letras maiúsculas e minúsculas.
 - Os nomes não podem incluir espaço em branco.

```
int studentAge = 20;  
String myCatchPhrase = "Aproveite o Dia de Avaliação  
de Alex!";
```

Encontre o valor de x

```
int x = 20;  
x = 25;  
x = 5 + 3;  
  
System.out.println(x);
```

- x é sempre igual a 20 ...
 - Até você atribuir outro valor a x.
- É possível atribuir a x um valor calculado.

Valores de x: ~~20~~ ~~25~~ 8

Encontre o valor de x

```
int x = 20;  
x = 25;  
x = 5 + 3;  
x = x + 1;  
x += 1;  
x++;  
System.out.println(x);
```

- É possível atribuir a `x` um novo valor com base em seu valor atual:
 - O Java fornece o operador `+=` abreviado para fazer isso.
 - Somar 1 a uma variável é tão comum que o Java fornece o operador `++` abreviado.

Valores de `x`: ~~20~~ ~~25~~ ~~8~~ ~~9~~ ~~10~~ 11

Encontre o valor de x novamente

- É possível atribuir a `x` o valor de outra variável:
 - Mudar `y` não muda `x`.
 - `y` e `x` são variáveis separadas.

```
int y = 20;  
int x = y;  
y++;  
  
System.out.println(x);  
System.out.println(y);
```

- Saída:

```
x 20  
y 21
```

Operadores Matemáticos Padrão

Finalidade	Operador	Exemplo	Comentários
Adição	+	<code>sum = num1 + num2;</code>	Se num1 é 10 e num2 é 2, sum é 12.
Subtração	-	<code>diff = num1 - num2;</code>	Se num1 é 10 e num2 é 2, diff é 8.
Multiplicação	*	<code>prod = num1 * num2;</code>	Se num1 é 10 e num2 é 2, prod é 20.
Divisão	/	<code>quot = num1 / num2;</code>	<p>Se num1 é 31 e num2 é 6, quot é 5.</p> <p>O resto é descartado.</p> <p>A divisão por 0 retorna um erro.</p>

Por quê?



Combinando Operadores para Fazer Atribuições

Finalidade	Operador	Exemplos <code>int a = 6, b = 2;</code>	Resultado
Somar a e atribuir	<code>+=</code>	<code>a += b</code>	<code>a = 8</code>
Subtrair de e atribuir	<code>-=</code>	<code>a -= b</code>	<code>a = 4</code>
Multiplicar por e atribuir	<code>*=</code>	<code>a *= b</code>	<code>a = 12</code>
Dividir por e atribuir	<code>/=</code>	<code>a /= b</code>	<code>a = 3</code>
Obter o resto e atribuir	<code>%=</code>	<code>a %= b</code>	<code>a = 0</code>

Operador de Módulo

Finalidade	Operador	Exemplo	Comentários
Resto	$\%$ / módulo	<pre>num1 = 31; num2 = 6; mod = num1 % num2; mod é 1</pre>	<p>Resto encontra o resto do primeiro número dividido pelo segundo.</p> <div><div>5 R 1</div><div>6 31 30 ---- 1</div></div> <p>Resto sempre fornece uma resposta com o mesmo sinal do primeiro operando.</p>

Operadores de Acréscimo e Decréscimo

(++ e --)

- A maneira longa:

`age = age + 1;`

ou

`count = count - 1;`

- A maneira curta:

`age++;`

ou

`count--;`

Mais sobre Operadores de Acréscimo e Decréscimo

Operador	Finalidade	Exemplo
++	Pré-acréscimo (++variável)	<pre>int id = 6; int newId = ++id; id é 7, newId é 7</pre>
	Pós-acréscimo (variável++)	<pre>int id = 6; int newId = id++; id é 7, newId é 6</pre>
--	Pré-decrécimo (--variável)	(O mesmo princípio se aplica.)
	Pós-decrécimo (variável--)	

Operadores de Acréscimo e Decréscimo

(++ e —)

```
1  int count=15;  
2  int a, b, c, d;  
3  a = count++;  
4  b = count;  
5  c = ++count;  
6  d = count;  
7  System.out.println(a + ", " + b + ", " + c + ", " + d);
```

Saída:

15, 16, 17, 17

Indução ao Erro na Divisão de Inteiros

- O animal comeu metade das galinhas.
- Quando dividimos nove galinhas por dois, o Java considera $9/2$ como 4.
 - Mas $9/2 = 4,5$.
 - O Java não deveria arredondar para 5?
 - O que está acontecendo aqui?



Divisão em Java

- Os números inteiros em Java não são arredondados.
- Os números inteiros em Java são **truncados**, o que significa que quaisquer números depois do ponto decimal serão removidos.

```
int x = 9/2;  
System.out.println(x);    //imprime 4
```

- Precisamos de outros tipos de dados se tivermos cenários que exijam uma precisão do ponto flutuante!

Tipos Primitivos de Ponto Flutuante

Tipo	Comprimento Flutuante	Quando usarei isso?
<code>float</code>	32 bits	Nunca
<code>double</code>	64 bits	Frequentemente

Dobra a precisão de um flutuante.

Exemplo:

```
public float pi    = 3.141592F;  
public double pi  = 3.141592;
```

Observe o F.

Indução ao Erro em double

- O problema original:

```
int x = 9/2;  
System.out.println(x);    //imprime 4
```

- double x deveria corrigir esse problema?

```
double x = 9/2;  
System.out.println(x);    //imprime 4,0
```

- Não?!?!
– Por que não?

Indução ao Erro em double

```
double x = 9/2;  
System.out.println(x);    //imprime 4,0
```

- O Java soluciona a expressão, trunca o 0,5 e transforma a resposta em um double.
- A expressão contém só valores inteiros. O Java não alocará a memória adicional necessária aos doubles até ele realmente precisar fazer isso.
- Solução: inclua double na expressão.

```
double x = 9/2,0;  
System.out.println(x);    //imprime 4,5
```

Uma Observação Final

- Declare uma variável com a palavra-chave `final` para fazer com que o valor dela seja inalterado (imutável).

```
final double PI = 3.141592;  
PI = 3.0;           //Não Permitido
```

- O Java impedirá se você tentar alterar o valor de uma variável final.
- Convenções de nomenclatura da variável final:
 - Capitalize todas as letras.
 - Use um caractere de sublinhado para separar as palavras.
 - `MINIMUM_AGE`
 - `SPEED_OF_LIGHT`

Parênteses em Expressões Matemáticas

- Esta expressão sem parênteses...

```
int x = 10 +20 +30 / 3;           //x=40
```

- é como escrever essa expressão com parênteses:

```
int x = 10 +20 +(30 / 3) ;        //x=40
```

- Se você quiser encontrar uma média, use parênteses como estes:

```
int x = (10 +20 +30) / 3;         //x=20
```

Precedência do Operador

- Veja aqui um exemplo da necessidade de regras de precedência:

```
int x = 25 - 5 * 4 / 2 - 10 + 4;
```

- Esta resposta é 34 ou 9?

Regras de Precedência

1. Operadores dentro de um par de parênteses
2. Operadores de acréscimo e decréscimo (++ ou --)
3. Operadores de multiplicação e divisão, calculados da esquerda para a direita
4. Operadores de adição e subtração, calculados da esquerda para a direita

Se operadores da mesma precedência aparecerem sucessivamente, eles serão avaliados da esquerda para a direita.

Usando Parênteses

- As expressões são avaliadas com as regras de precedência.
- No entanto, você deve usar parênteses para fornecer a estrutura pretendida.

Exemplos:

```
int x = ((25 - 5) * 4) / (2 - 10) + 4;  
int x = (20 * 4) / (2 - 10) + 4;  
int x = (80 / (2 - 10)) + 4;  
int x = (80 / -8) + 4;  
int x = -10 + 4;  
int x = -6;
```

Tipo Primitivo Textual

- O único tipo de dados textual primitivo é `char`.
- Ele é usado para um único caractere (16 bits).

Exemplo:

```
- char shirtSize = 'M';
```



As aspas simples devem ser usadas com valores literais de caractere.

Unindo Caracteres

Você pode usar caracteres juntos para criar frases.

- Esta é uma maneira ineficiente de fazer isso.
- É preciso haver uma linha de código para cada letra em uma sentença.

```
char letter1 = 'H';  
char letter2 = 'e';  
char letter3 = 'l';  
char letter4 = 'l';  
char letter5 = 'o';  
//É difícil codificar sentenças longas  
System.out.println(letter1 +letter2 +letter3  
                    +letter4 +letter5);
```

Unindo Caracteres de Forma Eficiente

Esta é uma maneira melhor

- Só é necessária uma linha para a sentença inteira:

```
String greeting = "Hello World!";  
//Observe as aspas duplas  
System.out.println(greeting);
```

Caracteres x Strings

- `char`s são para um único caractere.

– Use aspas simples.



```
char shirt1Size = 'S';  
char shirt2Size = 'M';  
char shirt3Size = 'L';
```

- `char`s não pode tratar vários caracteres.



```
char shirt4Size = 'XL';  
char shirt5Size = "XXL";
```


Caracteres x Strings

- Uma String pode tratar vários caracteres.
 - Use aspas duplas.



```
String shirt6Size = "XXXL";
```

Primitivas

Tipo	Comprimento	Dados
<code>boolean</code>	1 bit	<code>true</code> / <code>false</code>
<code>byte</code>	8 bits	Valores inteiros
<code>short</code>	16 bits	Valores inteiros
<code>int</code>	32 bits	Valores inteiros
<code>long</code>	64 bits	Valores inteiros
<code>float</code>	32 bits	Números de pontos flutuantes
<code>double</code>	64 bits	Números de pontos flutuantes
<code>char</code>	16 bits	Caracteres individuais

Onde estão as Strings?

Vamos investigar

- Podemos identificar outras diferenças entre `char` e `String`?

```
char shirt3Size = 'L';  
String shirt6Size = "XXXL";
```

1. `char` fica azul.

- `char` é uma palavra-chave de um tipo de dados primitivo.
- As palavras-chave ficam azuis no NetBeans

2. `String` é capitalizada.

- As strings são um objeto, e não uma primitiva.
- Por convenção, os tipos de objetos são capitalizados.

Strings São Objetos

- O Java vem com uma classe String que fornece detalhes sobre o seguinte:
 - Propriedades da string
 - Comportamentos da string
- As strings são objetos especiais.
 - As strings são tratadas de maneira um pouco diferente que a maioria dos objetos.
- Abordaremos mais estes pontos nas próximas seções:
 - Os objetos podem ter primitivas como propriedades.
 - Os objetos podem ter objetos como propriedades, como Strings.
 - Os objetos são armazenados de maneira diferente das primitivas na memória.

Declaração e Inicialização de Strings

Declare e atribua valores de `String` como faria com qualquer outra primitiva.

```
//Uma variável declarada e inicializada
int intVar = 300;
String stringVar = "Trezentos";

//Muitas variáveis declaradas
int x, y, z;
String xString, yString, zString;

//Uma variável declarada é inicializada
mais tarde
x = 1;
xString = "Um";
```

Variável da String x Literal da String

```
String stringVariable = "Esta é uma literal de String".;
```

Variável *Literal*

- É possível criar uma String combinando literais de Strings:

```
String combinedLiterals = "Quero" + " comprar uma camisa".;
```

- É possível criar uma String combinando variáveis de Strings:

```
String var1 = "Esta camisa tem";  
String var2 = " muitos botões";  
String combinedVariables = var1 + var2;
```

Concatenação de Strings

- A combinação de várias Strings é denominada concatenação.
- As strings não podem ser combinadas usando o operador +.
 - stringVariable1 + stringVariable2
 - stringVariable1 + "String literal"
 - stringVariable1 + "String literal" + stringVariable2

```
String greet1 = "Hello";  
String greet2 = "World";  
String message1 = greet1 + " " +greet2 + "!";  
String message2 = greet1 + " " +greet2 + " " +2016 + "!";
```

Saída de Concatenação da String

- Exemplo de concatenação:

```
String greet1 = "Hello";  
String greet2 = "World";  
String message1 = greet1 + " " +greet2 +"!";  
String message2 = greet1 + " " +greet2 +" " +2016 +"!";
```

- Você pode concatenar Strings dentro de uma instrução de impressão:

```
System.out.println(message2) ;  
System.out.println(greet1 + " " +greet2 +" " +2016 +"!") ;
```

Saída:

Hello World 2016!

Hello World 2016!

Combinando Strings e Números

- As strings podem conter números:

```
String totalPrice = "Total: $" +3;  
System.out.println(totalPrice);           //Total: $3
```

- Cuidado quando tentar fazer cálculos:

```
String totalPrice = "Total: $" +3 +2 +1;  
System.out.println(totalPrice);           //Total: $321
```


- Use parênteses para os números:

```
String totalPrice = "Total: $" +(3 +2 +1);  
System.out.println(totalPrice);           //Total: $6
```

Caracteres Especiais nas Strings

- Você lembra de quando imprimimos o gato?
- Na verdade, as duas barras invertidas não foram impressas:
 - Só uma barra invertida foi impressa.
 - Por quê?

```
3 public class Text01 {  
4     public static void main(String[] args) {  
5         System.out.println("  /\\"      /\\"  ");  
6         System.out.println(" /  \\"      /\\"  ");  
7         System.out.println(" /              \\" );  
8         System.out.println("(  /\\"      /\\"  )");  
9         System.out.println("====      V      =====");  
10        System.out.println("===== ( _ | _ ) =====");  
11        System.out.println("      (              )  ");  
12        System.out.println("      (              )  ");  
13    }  
14 }
```



Sequência de Escape

- Um caractere precedido de uma barra invertida denomina-se **sequência de escape** e tem um significado especial para o compilador.
- A tabela no próximo slide mostra as sequências de escape do Java.

Sequência de Escape

Sequência de Escape	Descrição
<code>\t</code>	Insere uma nova tabulação.
<code>\b</code>	Insere um caractere de retorno.
<code>\n</code>	Insere uma nova linha.
<code>\r</code>	Insere um retorno de carro.
<code>\f</code>	Insere um avanço de página.
<code>\'</code>	Insere um caractere de aspas simples.
<code>\"</code>	Insere um caractere de aspas duplas.
<code>\\</code>	Insere um caractere de barra invertida.

Sequência de Escape: Exemplo

Se você quiser inserir aspas dentro de aspas, deverá usar a sequência de escape, `\`, nas aspas internas.

– Para imprimir a sequência...

```
O gato fez "Miau!" para mim.
```

– Você escreveria

```
System.out.println("O gato fez \"Miau!\" para mim.");
```

Instruções de Impressão

- Escrever um texto em uma nova linha pode não imprimi-lo em uma nova linha:


```
System.out.println("Esta é a primeira linha."  
    + "Esta NÃO é a segunda linha.");
```

Saída:

Esta é a primeira linha. Esta NÃO é a segunda linha.

- As sequências de escape podem ajudar a formatar sua saída:

```
System.out.println("Esta é a primeira linha. \n"  
    + "Esta é a segunda linha.");
```



Saída:

Esta é a primeira linha.
Esta é a segunda linha.

Mais Instruções de Impressão

- Existem dois métodos importantes para impressão:

```
System.out.println("Método Imprimir linha");  
System.out.print("Método Imprimir");
```

- `println` funciona como se você estivesse inserindo `\n` no fim da instrução.
- As duas instruções a seguir produzem resultados equivalentes:

```
System.out.println("Imprimindo ");  
System.out.print("Imprimindo \n");
```

println() x print()

- **println()** cria automaticamente uma linha:

```
System.out.println("Esta é a primeira linha.");  
System.out.println("Esta é a segunda linha.");
```

Saída:

Esta é a primeira linha.

Esta é a segunda linha.

- **print()** não cria automaticamente uma linha:

```
System.out.print("Esta é a primeira linha.");  
System.out.print("Esta NÃO é a segunda linha.");
```

Saída:

Esta é a primeira linha. Esta NÃO é a segunda linha.

Atalho do NetBeans

Método Imprimir	Com que Frequência Usarei Esse Método?
<code>System.out.println()</code>	Frequentemente
<code>System.out.print()</code>	Raramente

- `System.out.println()` é usado com bastante frequência.
- Mas requer muita digitação para configuração.
- O Netbeans oferece um atalho:
 1. Digite `sout`.

```
sout
```

2. Pressione Tab.

```
System.out.println("");
```

Imprimir um Grande Volume de Texto, Opção 1

- Dependendo do que você esteja tentando imprimir, pode ser que prefira:
 - Dividir uma única instrução de impressão em muitas linhas no NetBeans:

```
System.out.println("Esta é a primeira linha."  
    + "Esta ainda é a primeira linha."  
    + "É apenas uma linha muito longa "  
    + "e eu não posso ver isso tudo no NetBeans."  
    + "\n" + "Esta é a segunda linha."  
    + "\n" + "Esta é a terceira linha.");
```

– OU...

Imprimir um Grande Volume de Texto, Opção 2

- Usar muitas instruções de impressão:

```
System.out.println("Esta é a primeira linha.");  
System.out.println("Esta é a segunda linha.");  
System.out.println("Esta é a terceira linha.");  
System.out.println("Esta é a quarta linha.");
```

Indução ao Erro em double

- O que já vimos:

```
double x = 9/2;    //Deve ser 4,5  
System.out.println(x);    //imprime 4
```

- O Java soluciona a expressão, trunca o 0,5 e, *em seguida*, transforma a resposta em um double.

- Simplificando o cenário, vemos:

```
double x = 4;  
System.out.println(x);    //imprime 4
```

- Estamos atribuindo um valor inteiro a uma variável dupla.
- O Java **promove** o valor inteiro para um duplo.

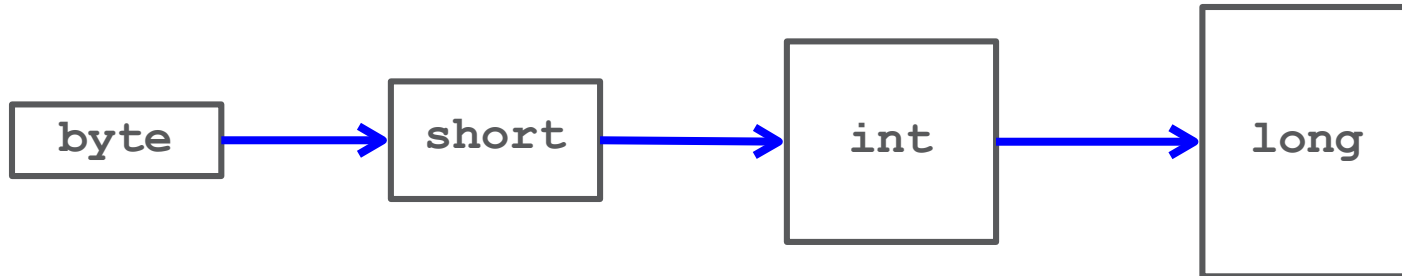

32 bits


64 bits

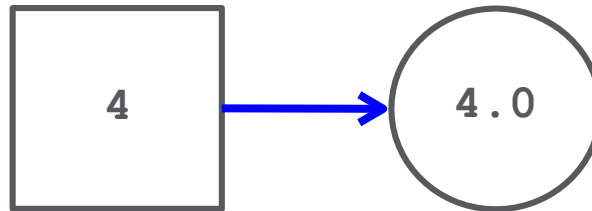
Promoção

- Promoções automáticas:

- Se você atribuir um tipo pequeno a um tipo maior:



- Se você atribuir um valor inteiro a um tipo de ponto flutuante:

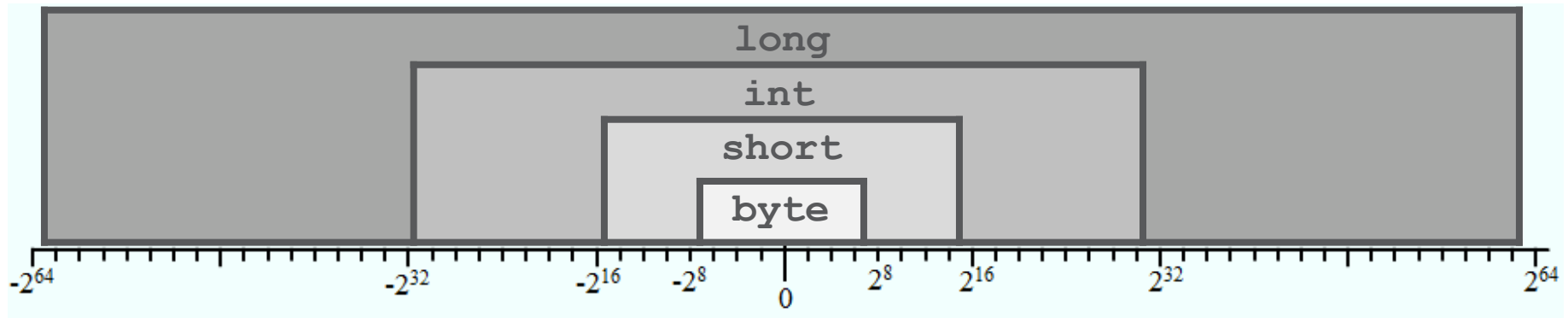


- Exemplos de promoções automáticas:

- `long intToLong = 6;`

- `double intToDouble = 4;`

Por que a Promoção Funciona?



- Um `byte` poderia variar de -128 a 127.
- Todos os valores `byte` possíveis podem estar contidos em `short`.
- Todos os valores `short` possíveis podem estar contidos em `int`.
- Todos os valores `int` possíveis podem estar contidos em `long`.
- Todos os valores `int` possíveis podem estar contidos em `double` sem perder a precisão.

Cuidado com a Promoção, Exemplo 1

- Equação: $55555 * 66666 = 3703629630$
- Exemplo de um possível problema:

```
int num1 = 55555;  
int num2 = 66666;  
long num3;  
num3 = num1 * num2;
```

- Exemplo de uma possível solução:

```
int num1 = 55555;  
long num2 = 66666; ———— Alterado de int para long  
long num3;  
num3 = num1 * num2;
```

Cuidado com a Promoção, Exemplo 2

- Equação: $7 / 2 = 3.5$
- Exemplo de um possível problema:

```
int num1 = 7;  
int num2 = 2;  
double num3;  
num3 = num1 / num2;           //num3 é 3
```

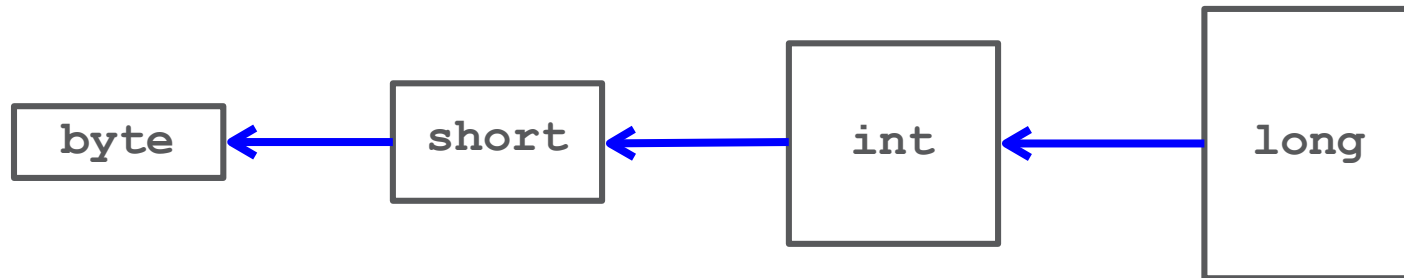
- Exemplo de uma possível solução:

```
int num1 = 7;  
double num2 = 2;           ————Alterado de int para double  
double num3;  
num3 = num1 / num2;           //num3 é 3,5
```

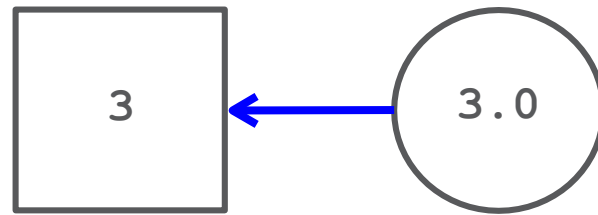

Conversão de Tipo

- Quando converter:

- Se você atribuir um tipo maior a um tipo menor:



- Se você atribuir um tipo de ponto flutuante a um tipo integral:



- Exemplos de conversão:

- `int longToInt = (int) 20L;`
- `short doubleToShort = (short) 3.0;`

Cuidado com a Conversão de Tipo

- Esteja ciente sobre a perda de precisão.
- Exemplo de um possível problema:

```
int myInt;  
double myPercent = 51.9;  
myInt = (int)myPercent; // O número é "cortado"  
                        // myInt é 51
```

Cuidado com a Conversão de Tipo

- Exemplo de um possível problema:

```
int myInt;  
long myLong = 123987654321L;  
myInt = (int)myLong;    // O número é "cortado"  
                        // myInt é -566397263
```

- Exemplo mais seguro de conversão:

```
int myInt;  
long myLong = 99L;  
myInt = (int)myLong;    // Não há perda de dados;  
                        // só zeros.  
                        // myInt é 99
```

Suposições do Compilador para Tipos de Dados Integral e de Ponto Flutuante

- A maioria das operações resulta em `int` ou `long`.
 - Os valores `byte`, `short` e `char` são promovidos automaticamente para `int` antes de uma operação.
 - Se uma expressão contiver `long`, toda ela será promovida para longa.
- Se uma expressão contiver um tipo flutuante, toda ela será promovida para um tipo flutuante.
- Todos os valores de pontos flutuantes literais são vistos como `double`.

Opções para Corrigir Problemas

Exemplo de um possível problema:

```
int num1 = 53; // 32 bits de memória para conter o valor
int num2 = 47; // 32 bits de memória para conter o valor
byte num3;      // 8 bits de memória reservados
num3 = (num1 + num2); // causa erro do compilador
```

- Um `byte` deve ser capaz de conter o valor 100.
- Mas o Java recusa-se a fazer a atribuição e emite um erro de "possível perda de precisão".
- O Java considera que adicionar variáveis `int` resultará em um valor que estouraria o espaço alocado para um `byte`.

Opções para Corrigir Problemas

- Solução usando um tipo de dados maior:

```
int num1 = 53;  
int num2 = 47;  
int num3;      ———— Alterado de byte para int  
num3 = (num1 + num2);
```

- Solução usando a conversão:

```
int num1 = 53; // 32 bits de memória para conter o valor  
int num2 = 47; // 32 bits de memória para conter o valor  
byte num3;     // 8 bits de memória reservados  
num3 = (byte) (num1 + num2); // não há perda de dados
```

Promoção Automática

- Exemplo de um possível problema:

```
short a, b, c;  
a = 1 ;  
b = 2 ;  
c = a + b ; //erro do compilador
```

a e b são promovidos automaticamente a
números inteiros

- Exemplo de possíveis soluções:
 - Declare c como um tipo `int` na declaração original:
`int c;`
 - Converta o tipo do resultado (`a+b`) na linha de atribuição:
`c = (short) (a+b) ;`

Usando um Longo

```
public class Person {
```

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

```
        int ageYears = 32;
```

```
        int ageDays = ageYears * 365;
```

```
        long ageSeconds = ageYears * 365 * 24L * 60 * 60;
```


```
        System.out.println("Você já tem " + ageDays + " dias de idade.");
```

```
        System.out.println("Você já tem " + ageSeconds + " segundos de idade.");
```

```
    } // fim do método main
```

```
} // fim da classe
```

O uso do L para indicar um longo fará com que o compilador reconheça o resultado total como um longo.



Usando Pontos Flutuantes

As expressões são promovidas automaticamente a pontos flutuantes.

- Exemplo de possível problema:

```
int num1 = 1 + 2 + 3 + 4.0;
int num2 = (1 + 2 + 3 + 4) * 1.0;
```

//erro do compilador
//erro do compilador

- Exemplo de possíveis soluções:

– Declare `num1` e `num2` como tipos `double`:

```
double num1 = 1 + 2 + 3 + 4.0;           //10
double num2 = (1 + 2 + 3 + 4) * 1.0;      //10
```

– Converta `num1` e `num2` como tipos `int` na linha de atribuição:

```
int num1 = (int) (1 + 2 + 3 + 4.0);       //10
int num2 = (int) ((1 + 2 + 3 + 4) * 1.0); //10
```

Atribuição e Tipos de Dados de Ponto Flutuante

- Exemplo de possível problema:

```
float float1 = 27.9;    //erro do compilador
```

- Exemplo de possíveis soluções:

- O F avisa ao compilador que 27,9 é um valor `float`:

```
float float1 = 27.9F;
```

- 27,9 é conversão para um tipo `float`:

```
float float1 = (float) 27.9;
```

O Sublinhado

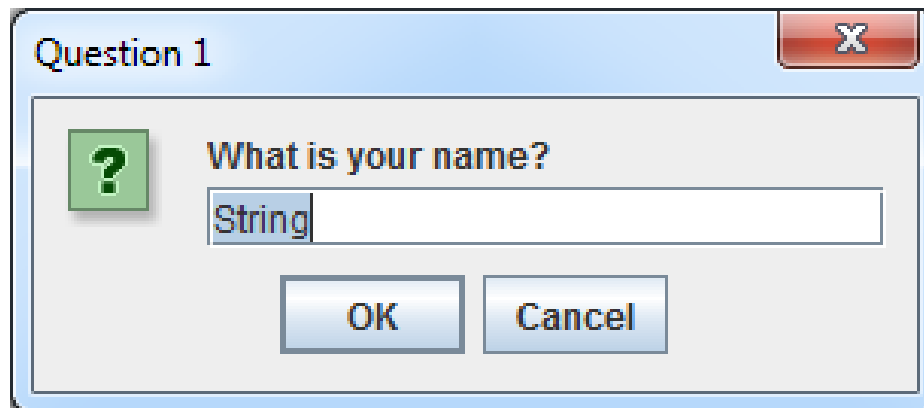
- Você deve ter percebido o uso de sublinhados (_):
 - A partir do Java SE7, você pode incluir sublinhados ao atribuir valores numéricos.
 - Os sublinhados ajudam a tornar os números grandes mais legíveis.
 - Os sublinhados não afetam o valor de uma variável.
- As duas instruções a seguir são equivalentes:

```
int x = 123_456_789;
```

```
int x = 123456789;
```

Convertendo Strings em Dados Numéricos

- Quando você convida um usuário a digitar em uma caixa de diálogo...
 - Ele pode digitar o que quiser.
 - Este texto é melhor representado por uma `String`.
- Às vezes, você precisará fazer cálculos com entradas de usuário.
 - Se projetar um programa que aceite entrada de texto, pode ser que você precise converter a `String` em um tipo de dados numérico.



Fazendo Parse de Strings

A conversão de texto em dados numéricos é uma forma de parse.

- Como converter uma `String` em `int`:

```
int intVar1 = Integer.parseInt("100");
```

- Como converter uma `String` em `double`:

```
double doubleVar2 = Double.parseDouble("2.72");
```

Problema com Entrada de Usuário

- `NumberFormatException`
 - Ocorre porque não é possível fazer parse de um valor.
 - É arriscado quando os usuários podem inserir qualquer valor que desejarem.



```
int intVar1 = Integer.parseInt("Puppies!");
```

- O software não deve travar devido a uma entrada do usuário.
 - Mas ignore isso por enquanto.
 - Primeiro, vamos imaginar como obter a entrada do usuário na próxima lição.
 - Aprenderemos a tratar erros e exceções na Seção 8.

Por Que Você Deve Obter uma Entrada do Usuário?

- Quando você atribui manualmente valores a variáveis, esse procedimento é conhecido como **hard-coding** de valores:

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

- É possível alterar facilmente valores submetidos a hard-code porque você tem o código-fonte e o NetBeans:

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

- Mas, quando você distribui um software, seus usuários não podem se dar a esse luxo.

Tipos de Entrada de Usuário

- Exemplos de entrada de usuário incluem o seguinte...
 - Pressionar um botão em um controlador de jogo
 - Inserir um endereço em um GPS
 - Inserir números e funções em uma calculadora
 - Informar seu nome às pessoas
- Mas sem entrada do usuário...
 - Quando o jogo fará com que seu personagem pule?
 - Aonde seu GPS guiará você?
 - Que números sua calculadora tritulará?
 - Do que as pessoas chamarão você?

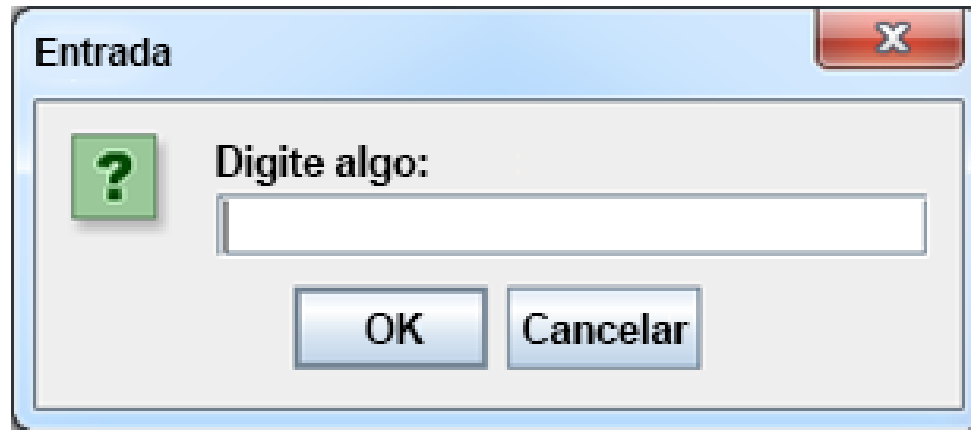
Como Obter a Entrada do Usuário

- 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`
 - **JavaFX** (um sucessor do Swing, abordado mais adiante)
 - `Scanner`

JOptionPane

Essa é uma maneira simples de obter entrada dos usuários:

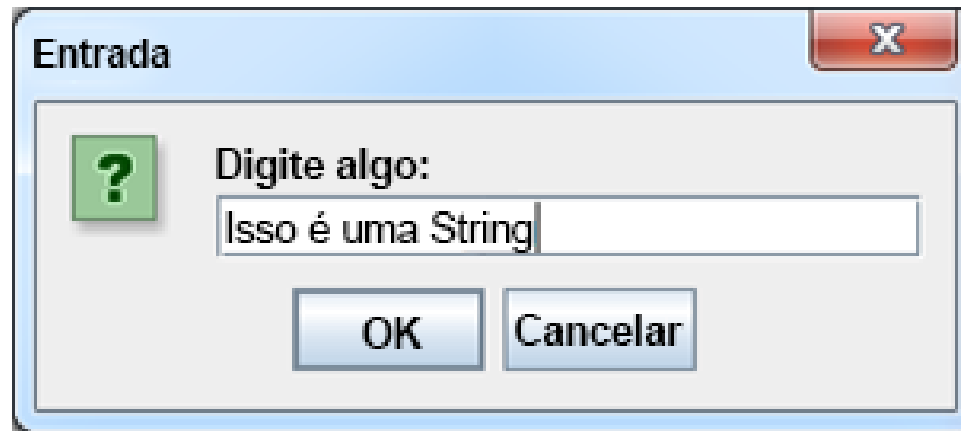
```
JOptionPane.showInputDialog("Digite algo:");
```



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++;
```

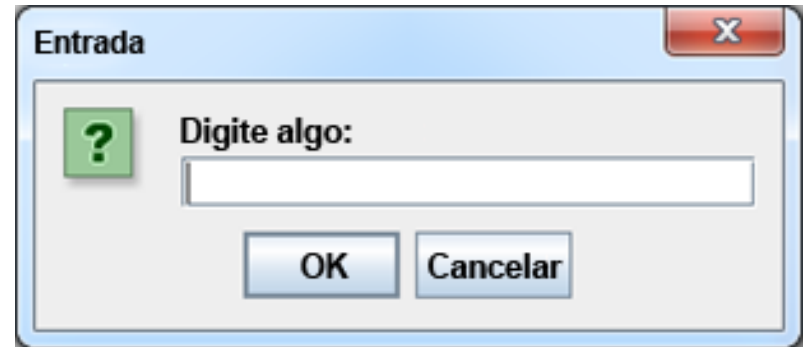
- Ou condensá-la em uma única linha:

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

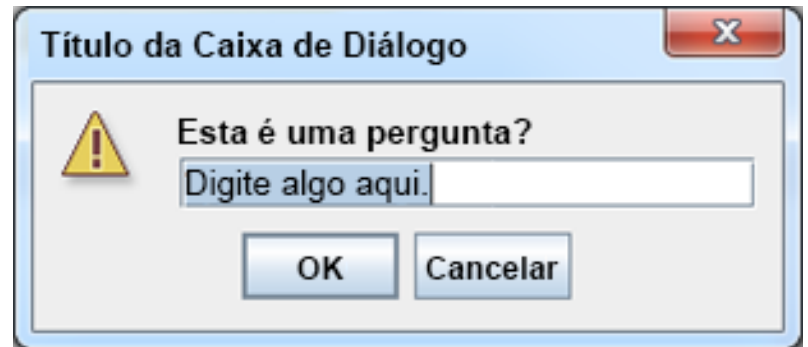
- Essa opção é uma questão de preferência pessoal.
 - Mas, se você precisar fazer referência a determinados valores novamente mais tarde, convém armazená-los em uma variável.

InputDialogs (caixas de diálogo de entrada) Diferentes

Criamos uma InputDialog simples:

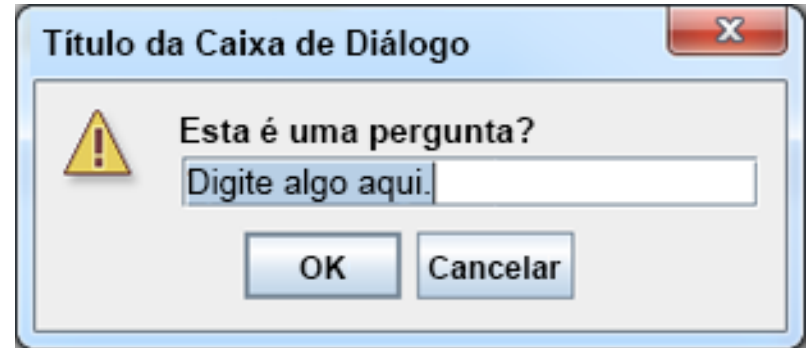


Com um código mais complexo, podemos personalizar mais a InputDialog:



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:

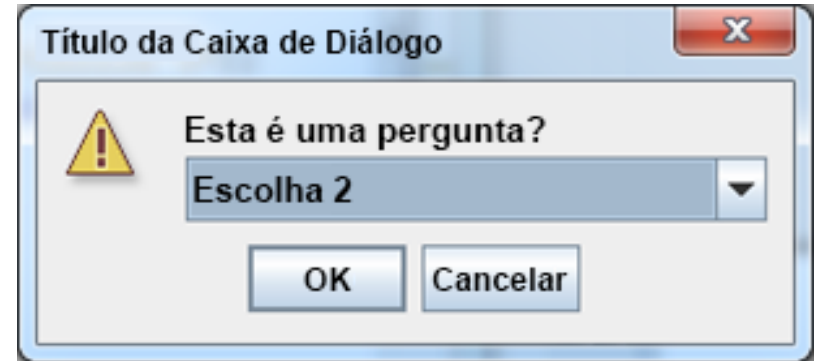


Conversão

```
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

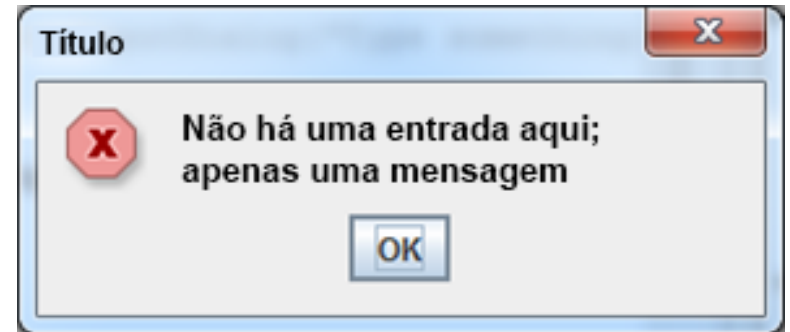
- Para evitar uma entrada indesejada, é possível fornecer somente valores aceitáveis para os usuários.
- Parte dessa sintaxe será discutida em detalhes na Seção 8.



```
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);
```

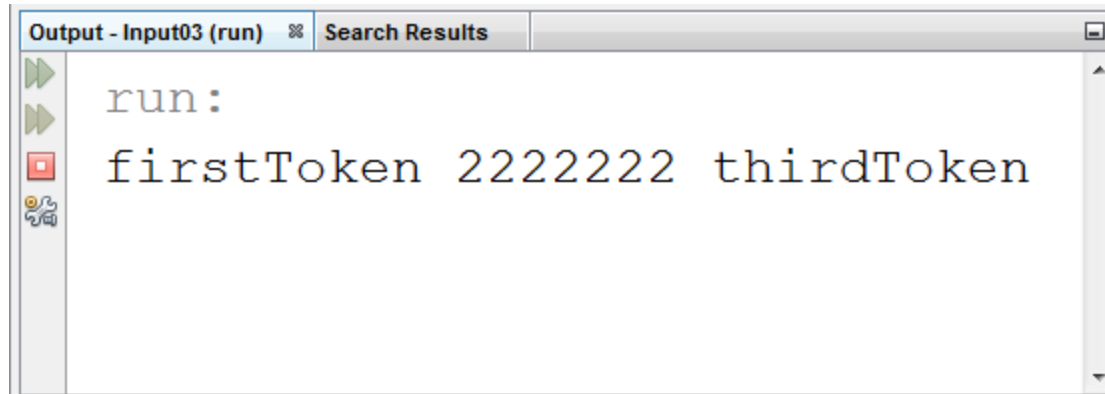

Obtendo Entrada com um Scanner

- 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
```

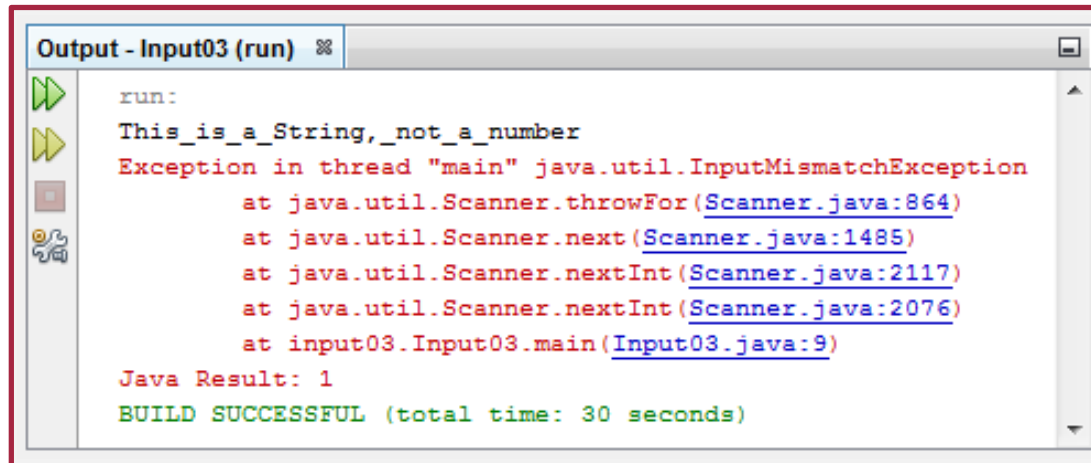
The text is displayed in a monospaced font. The output window has a standard IDE interface with a title bar, a search bar, and a vertical scrollbar on the right.

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

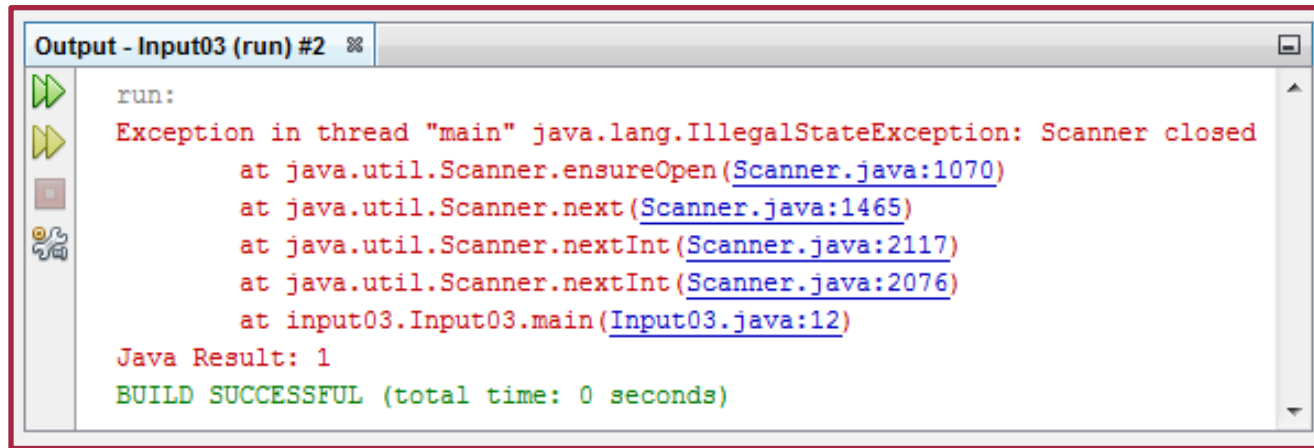
A screenshot of an IDE's output window titled "Output - Input03 (run)". The window shows the execution of a Java program. It starts with "run:" followed by the input "This_is_a_String,_not_a_number". Then, it displays a red error message: "Exception in thread 'main' java.util.InputMismatchException". 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 entries 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

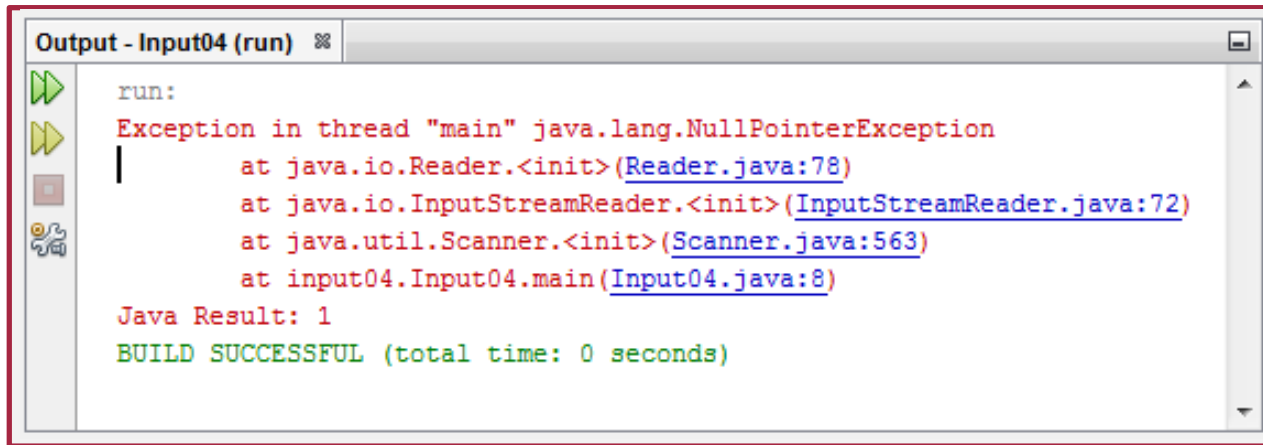
A screenshot of an IDE's output window titled "Output - Input03 (run) #2". The window shows the execution of a Java program. It starts with "run:" followed by a red error message: "Exception in thread \"main\" java.lang.IllegalStateException: Scanner closed". Below this, the stack trace is listed in 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 error propagated through Scanner.java:563, InputStreamReader.java:72, and Reader.java:78. The program result is 1, and the build was successful.

```
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();  
}
```