



Cadeias de caracteres

- Em Java (e LOOs em geral) cadeias de caracteres são objetos, da classe *String*.
- Literal entre aspas duplas (ex: `"a3*\n"`) cria objeto.
- Posição do primeiro caractere, em objeto do tipo *String*, é igual a zero, do segundo igual a um, e assim por diante.

Operações sobre cadeias de caracteres

| | |
|--|---------------|
| Indexação | <i>charAt</i> |
| Tamanho (n ^o de caracteres) | <i>length</i> |

Sendo s expressão que representa objeto da classe *String*,
 e expressão que denota valor inteiro não-negativo n :

$s.charAt(e)$

retorna caractere na posição n de s

Operações sobre cadeias de caracteres

| | |
|--|---------------|
| Indexação | <i>charAt</i> |
| Tamanho (n ^o de caracteres) | <i>length</i> |

Sendo s expressão que representa objeto da classe *String*,
 e expressão que denota valor inteiro não-negativo n :

$s.charAt(e)$

"xpto".*charAt*(0)

retorna caractere na posição n de s

Operações sobre cadeias de caracteres

| | |
|--|---------------|
| Indexação | <i>charAt</i> |
| Tamanho (n ^o de caracteres) | <i>length</i> |

Sendo s expressão que representa objeto da classe *String*,
 e expressão que denota valor inteiro não-negativo n :

$s.charAt(e)$

| | |
|---------------------------|-----|
| "xpto". <i>charAt</i> (0) | 'x' |
| "xpto". <i>charAt</i> (3) | |

retorna caractere na posição n de s

Operações sobre cadeias de caracteres

| | |
|--|---------------|
| Indexação | <i>charAt</i> |
| Tamanho (n ^o de caracteres) | <i>length</i> |

Sendo s expressão que representa objeto da classe *String*,
 e expressão que denota valor inteiro não-negativo n :

$s.charAt(e)$

retorna caractere na posição n de s

| | |
|---------------------------|-----|
| "xpto". <i>charAt</i> (0) | 'x' |
| "xpto". <i>charAt</i> (3) | 'o' |
| "xpto". <i>length</i> () | |

Operações sobre cadeias de caracteres

| | |
|--|---------------|
| Indexação | <i>charAt</i> |
| Tamanho (n ^o de caracteres) | <i>length</i> |

Sendo s expressão que representa objeto da classe *String*,
 e expressão que denota valor inteiro não-negativo n :

$s.charAt(e)$

retorna caractere na posição n de s

| | |
|---------------------------|-----|
| "xpto". <i>charAt</i> (0) | 'x' |
| "xpto". <i>charAt</i> (3) | 'o' |
| "xpto". <i>length</i> () | 4 |

Classes invólucros

- Todo programa Java importa automaticamente a biblioteca *java.lang*.
- Essa biblioteca contém (dentre outras) as chamadas “classes invólucros”:

| | | |
|----------------|----------------|------------------|
| <i>Boolean</i> | <i>Byte</i> | <i>Character</i> |
| <i>Short</i> | <i>Integer</i> | <i>Long</i> |
| <i>Double</i> | <i>Float</i> | |

Conversão de cadeia de caracteres em valor básico

| Para | Método | Exemplo de expressão |
|-------------|--|--|
| int | <i>parseInt</i> | <i>Integer.parseInt(s)</i> |
| long | <i>parseLong</i> | <i>Long.parseLong(s)</i> |
| float | <i>parseFloat</i> | <i>Float.parseFloat(s)</i> |
| double | <i>parseDouble</i> | <i>Double.parseDouble(s)</i> |
| boolean | <i>valueOf,</i> <i>booleanValue</i> | <i>Boolean.valueOf(s).booleanValue()</i> |

De cadeia de caracteres para objeto de classe invólucro

Seja C classe invólucro e s expressão do tipo *String*.

```
C.valueOf(s)
```

cria objeto da classe C e armazena nesse objeto resultado da conversão de cadeia de caracteres s para tipo básico correspondente (ou causa a exceção *NumberFormatException*, caso a cadeia de caracteres não represente um valor do tipo desejado).

Ex: *Integer.valueOf(s)* retorna um valor igual ao fornecido por:

```
new Integer(Integer.parseInt(s))
```

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

```
Integer.toString(123)
```

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | |

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | "0.1" |
| <i>Float.toString</i> (1e-1f) | |

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | "0.1" |
| <i>Float.toString</i> (1e-1f) | "0.1" |

- Método sobrecarregado *valueOf* da classe *String*:

| | |
|----------------------------|--|
| <i>String.valueOf</i> (10) | |
|----------------------------|--|

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | "0.1" |
| <i>Float.toString</i> (1e-1f) | "0.1" |

- Método sobrecarregado *valueOf* da classe *String*:

| | |
|-----------------------------|------|
| <i>String.valueOf</i> (10) | "10" |
| <i>String.valueOf</i> (0.1) | |

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | "0.1" |
| <i>Float.toString</i> (1e-1f) | "0.1" |

- Método sobrecarregado *valueOf* da classe *String*:

| | |
|-------------------------------|-------|
| <i>String.valueOf</i> (10) | "10" |
| <i>String.valueOf</i> (0.1) | "0.1" |
| <i>String.valueOf</i> (1e-1f) | |

Conversão para cadeia de caracteres

- método estático *toString* de classes invólucros

| | |
|-------------------------------|-------|
| <i>Integer.toString</i> (123) | "123" |
| <i>Double.toString</i> (0.1) | "0.1" |
| <i>Float.toString</i> (1e-1f) | "0.1" |

- Método sobrecarregado *valueOf* da classe *String*:

| | |
|-------------------------------|-------|
| <i>String.valueOf</i> (10) | "10" |
| <i>String.valueOf</i> (0.1) | "0.1" |
| <i>String.valueOf</i> (1e-1f) | "0.1" |

Classe *Character*

```
public static boolean isDigit (char c)
public static boolean isLetter (char c)
public static boolean isLetterOrDigit (char c)
public static boolean isLowerCase (char c)
public static boolean isUpperCase (char c)
public static boolean isSpace (char c)
public static char toLowerCase (char c)
public static char toUpperCase (char c)
public char charValue ()
```

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

```
"abcd" + "ef"
```

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|---------------|----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | |

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|---------------|----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | "abcd1" |
| "abcd" + 1.0 | |

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|----------------|-----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | "abcd1" |
| "abcd" + 1.0 | "abcd1.0" |
| "abcd" + 1e-1f | |

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|----------------|-----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | "abcd1" |
| "abcd" + 1.0 | "abcd1.0" |
| "abcd" + 1e-1f | "abcd0.1" |
| "abcd" + 1 + 2 | |

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|----------------|-----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | "abcd1" |
| "abcd" + 1.0 | "abcd1.0" |
| "abcd" + 1e-1f | "abcd0.1" |
| "abcd" + 1 + 2 | "abcd12" |
| 1 + 2 + "abcd" | |

Concatenação de cadeias de caracteres

- Operador +
- Quando um dos argumentos é um valor básico chama implicitamente método *toString* depois de criar objeto da classe invólucro

| | |
|----------------|-----------|
| "abcd" + "ef" | "abcdef" |
| "abcd" + 1 | "abcd1" |
| "abcd" + 1.0 | "abcd1.0" |
| "abcd" + 1e-1f | "abcd0.1" |
| "abcd" + 1 + 2 | "abcd12" |
| 1 + 2 + "abcd" | "3abcd" |



Comparação de cadeias de caracteres

- Método *equals* testa igualdade de cadeias de caracteres (ou seja, dos caracteres componentes das cadeias).
- Ao contrário, `==` determina se objetos comparados são iguais (ou seja, se são o mesmo objeto).

Comparação de cadeias de caracteres

- Cada literal da classe *String* representa um dado objeto.
Ex: "abcd" == "abcd" retorna true (apesar de **primeiro** uso de aspas duplas significar criação de objeto da classe *String*).
- Para isso, a cada uso de um literal, é necessário determinar, em tempo de execução, se o objeto correspondente já foi anteriormente criado. Em caso positivo, uma referência ao objeto é retornada. Em caso negativo, um novo objeto é inserido no conjunto de objetos criados.

Comparação de cadeias de caracteres

```
"ab".equals("ab")
```

Comparação de cadeias de caracteres

| | |
|--------------------------------|-------------------|
| <code>"ab".equals("ab")</code> | <code>true</code> |
| <code>"ab" == "ab"</code> | |

Comparação de cadeias de caracteres

| | |
|---------------------------------------|-------------------|
| <code>"ab".equals("ab")</code> | <code>true</code> |
| <code>"ab" == "ab"</code> | <code>true</code> |
| <code>"ab" == new String("ab")</code> | |

Comparação de cadeias de caracteres

| | |
|---------------------------------------|--------------------|
| <code>"ab".<i>equals</i>("ab")</code> | <code>true</code> |
| <code>"ab" == "ab"</code> | <code>true</code> |
| <code>"ab" == new String("ab")</code> | <code>false</code> |

Conversão de Tipo

$(t) e$

- Converte expressão e para o tipo t , se possível
- Caso contrário, um erro é detectado (em geral durante a compilação)



Conversão de Tipo

- Tipos numéricos: **extensão** ou **truncamento**.
- **Extensão**: simples atribuição de valores apropriados aos bits adicionais da representação.
- **Truncamento** pode envolver mudança de valor.

Conversão de Tipo

- Extensões ocorrem implicitamente, sem nunca provocar ocorrência de erro.
- Uma extensão de t_e para t pode envolver dois tipos quaisquer nas seguintes cadeias (t_e precedendo t na cadeia);

```
byte--short--int--long--float--double  
char --int--long--float--double
```



Conversão de tipo

Porque existem duas cadeias de conversão implícita

byte–short–int–long–float–double

char –int–long–float–double

e não apenas uma ?



Conversão de tipo

Porque existem duas cadeias de conversão implícita

byte–short–int–long–float–double

char –int–long–float–double

e não apenas uma ?

Porque uma conversão implícita (uma extensão) nunca pode modificar o valor representado nem causar um erro; portanto, `byte` e `short`, que podem ser negativos, não podem ser convertidos implicitamente para `char`, que é sempre não-negativo.



Conversão de tipo: Certo ou Errado?

- byte $b = 128$;



Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;`

Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;` Certo.
- `byte a, b = 1; byte c = a+b;`

Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;` Certo.
- `byte a, b = 1; byte c = a+b;` Errado: `byte` é estendido para `int` para realização de `a+b`.
- `byte a, b = 1; byte c = (byte)(a+b);`

Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;` Certo.
- `byte a, b = 1; byte c = a+b;` Errado: `byte` é estendido para `int` para realização de `a+b`.
- `byte a, b = 1; byte c = (byte)(a+b);` Correto.
- `float a=1.0, byte b = a;`

Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;` Certo.
- `byte a, b = 1; byte c = a+b;` Errado: `byte` é estendido para `int` para realização de `a+b`.
- `byte a, b = 1; byte c = (byte)(a+b);` Correto.
- `float a=1.0, byte b = a;` Errado: em geral, perda de precisão possível.
- `float a=1.0, byte b = (byte)a;`

Conversão de tipo: Certo ou Errado?

- `byte b = 128;` Errado: 128 não pode ser representado em 1 byte (não pode ser convertido de `int` para `byte`).
- `byte b = -128;` Certo.
- `byte a, b = 1; byte c = a+b;` Errado: `byte` é estendido para `int` para realização de `a+b`.
- `byte a, b = 1; byte c = (byte)(a+b);` Correto.
- `float a=1.0, byte b = a;` Errado: em geral, perda de precisão possível.
- `float a=1.0, byte b = (byte)a;` Correto.



Entrada e Saída Textual: Primeiras Noções

- Operação de saída (ou escrita, ou gravação) de dados envia dados para um dispositivo de saída conectado ao computador (ex: tela de terminal, impressora, disco, fita etc.).
- Operação de entrada (ou leitura) de dados obtém dados de um dispositivo de entrada (ex: teclado, disco etc.).
- Dados armazenados como seqüências de bytes ou caracteres.
- Dados podem ser armazenados em **arquivos**, organizados em estrutura hierárquica de conjuntos de arquivos (chamados de **diretórios**) para facilitar localização e uso de arquivos.



Entrada e Saída Textual

- **Texto:** seqüência de *linhas*, separadas por caractere terminador de linha ('`\n`').
- Escrevendo caracteres em dispositivo de saída padrão:

System.out.println(e)

- ★ *out*: variável estática declarada na classe *System*.
- ★ tipo de *out*: *PrintStream*, contém definição do método *println*.

Entrada e Saída Textual

- Lendo caracteres do dispositivo de entrada padrão:

```
int a = System.in.read();
```

armazena em *a* um único byte lido do dispositivo de entrada padrão.

- Valor entre 0 e 255 representa código do caractere lido.

Entrada e saída textual

Leitura de um único caractere (representável em um byte), do dispositivo de entrada padrão, e escrita desse caractere lido no dispositivo de saída padrão:

```
import java.io.*;

class ESTextual
{ public static void main (String[] args) throws IOException
  { int a = System.in.read();
    System.out.println((char) a); }
}
```

`import java.io.*;` “importa” nomes definidos em “biblioteca” `java.io` (explicação detalhada mais à frente).

`throws IOException` também explicado mais adiante.

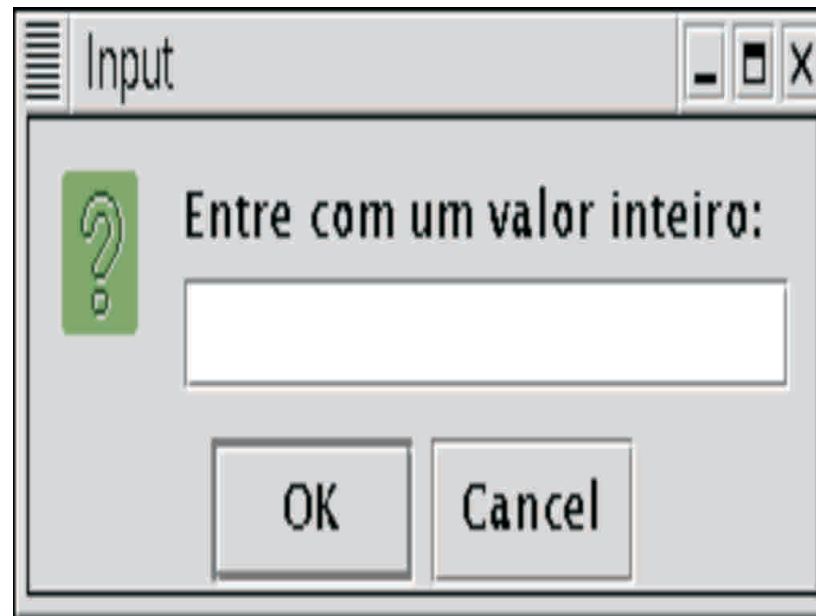




Entrada e Saída em Janelas, Campos de Texto e Botões

- E/S em componentes de “interface gráfica” (ou “interface visual”)
- Modelo de **tratamento de eventos**: ações (ex: pressionar botão do *mouse*) originam chamadas a métodos.
- Bibliotecas **AWT** e **Swing** permitem tratamento de eventos em componentes de interface gráfica

showInputDialog

```
String entrada = JOptionPane.showInputDialog  
    ("Entre com um valor inteiro:");
```





showInputDialog

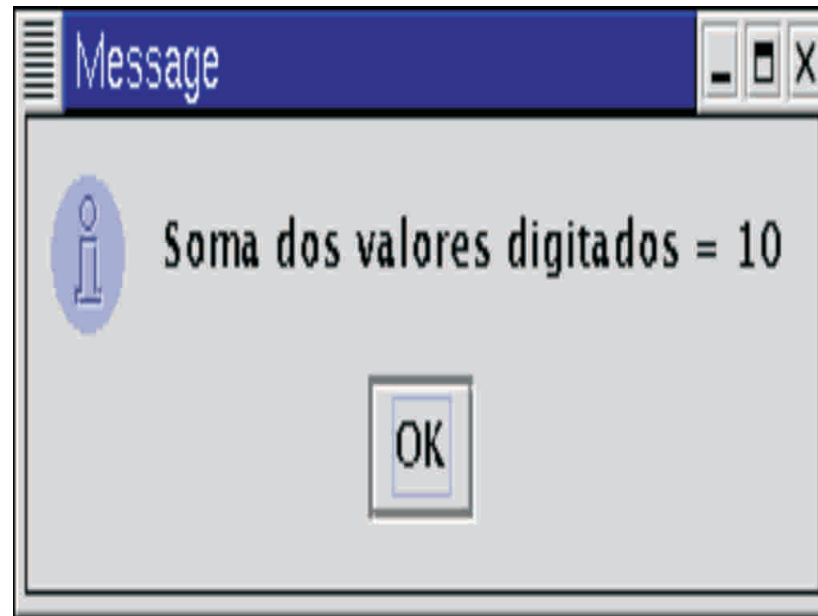
- *showInputDialog* cria janela e espera usuário digitar cadeia de caracteres no campo de texto.
- **Botão *OK* “clicado” ou tecla *Enter* pressionada:**
⇒ cadeia digitada é retornada como objeto da classe *String* (e janela desaparece).
- **Botão *Cancel* “clicado”:** ⇒ valor `null` é retornado.

showMessageDialog

```
import javax.swing.*;
class JanelasDeDialogo1
{ public static void main(String[] a)
  { String entrada = JOptionPane.showInputDialog
    ("Entre com um valor inteiro: ");
    int x = Integer.parseInt(entrada);
    entrada = JOptionPane.showInputDialog
    ("Entre com outro valor inteiro: ");
    int y = Integer.parseInt(entrada);

    JOptionPane.showMessageDialog
    (null, "Soma dos valores digitados = " + (x+y)); }
}
```

showMessageDialog





showMessageDialog

- 1^o argumento especifica componente abaixo do qual a janela deve ser criada
- se `null` (isto é, nenhum componente) for especificado, a janela é criada no centro da tela
- 2^o argumento especifica cadeia de caracteres a ser mostrada na janela