

# Tipos de Dados Fundamentais

Roland Teodorowitsch

Fundamentos de Programação - Escola Politécnica - PUCRS

15 de março de 2023

# Introdução

# Objetivos

- declarar e inicializar variáveis e constantes
- entender as propriedades e limitações de números inteiros e de ponto-flutuante
- apreciar a importância de comentários e bons leiautes de código
- escrever expressões aritméticas e comandos de atribuição
- criar programas que leiam e processem entradas, e exibam os resultados
- aprender a usar o tipo String de Java

# Conteúdos

- variáveis
- aritmética
- entrada e saída
- resolução de problemas: primeiro faça à mão
- cadeias de caracteres (*strings*)

# Variáveis

# Variáveis

- a maioria dos programas de computador armazena valores temporários em locais de armazenamento identificados
  - os programadores identificam estes locais para facilitar o acesso
- existem muitos tipos diferentes de armazenamento (com tamanhos diferentes) para guardar coisas diferentes
- você “declara” uma variável informando ao compilador:
  - qual o tipo/tamanho da variável que você precisa
  - por qual nome você quer se referir a ela

# Declaração de Variáveis

- quando se declara uma variável, você frequentemente especifica um valor inicial para ela
- isto também é onde você diz ao compilador o tipo/tamanho que ela poderá armazenar
- por exemplo:

```
int latasPorPacote = 6;
```

- `int` é um tipo para números inteiros com sinal
- `double` é um tipo para números de ponto-flutuante e `String` é um tipo para textos
- use nomes de variáveis (`latasPorPacote`) sugestivos
- a atribuição de um valor inicial é opcional, mas geralmente é uma boa ideia
- uma declaração de variáveis termina por ;

## Um exemplo: venda de refrigerantes

- Refrigerantes são vendidos em latas e garrafas. Uma loja oferece um pacote de 6 latas, cada lata com 355ml, pelo mesmo preço de uma garrafa de 2 litros. Qual você deve comprar? (1 litro = 1000 ml)
- Lista de variáveis:
  - número de latas por pacote (valor inteiro)
  - litros em uma garrafa (valor inteiro)
  - litros em uma lata (valor fracionário)





# Variáveis e Conteúdos

- Você pode (opcionalmente) definir o conteúdo de uma variável quando você declara ela

```
int latasPorPacote = 6;
```

- Uma variável é um local de armazenamento com um nome
- Imagine, por exemplo, uma vaga de estacionamento em um edifício-garagem:
  - Identificador: J053
  - Conteúdo: carro do João



# Exemplos de Declaração de Variáveis em Java

V/F	Código	Descrição
V	<code>int latas = 6;</code>	Declara uma variável inteira e inicializa ela com 6
V	<code>int total = latas + garrafas;</code>	O valor inicial não precisa ser um valor fixo. (Naturalmente, <code>latas</code> e <code>garrafas</code> devem ter sido declarados e inicializados.)
X	<code>garrafas = 1;</code>	<b>ERRO:</b> está faltando o tipo. Esta expressão não é uma declaração, mas a atribuição de um novo valor para a variável.
X	<code>int volume="2";</code>	<b>ERRO:</b> você não pode inicializar um número com um <i>string</i>
V	<code>int latasPorPacote;</code>	Declara uma variável inteira sem inicializá-la. Isto pode causar erros.
V	<code>int reais, centavos;</code>	Declara 2 variáveis inteiras em uma sentença.

# Por quê há diferentes tipos?

- Há 3 tipos diferentes de variáveis que usaremos neste capítulo
  - 1 um número inteiro (sem parte fracionária): **int**
  - 2 um número com parte fracionária: **double**
  - 3 uma palavra (um grupo de caracteres): **String**
- Especifique o tipo antes do nome nas declarações:

```
int latasPorPacote = 6;  
double volumeLata = 0.355;
```

# Por quê há diferentes tipos?

- De volta à analogia do edifício-garagem, existem espaços diferentes para diferentes tipos de veículos
  - bicicleta
  - motocicleta
  - carros grandes
  - carros compactos

# Constantes Numéricas em Java

- Algumas vezes quando você digita um número, o compilador tem que “adivinhar” de qual tipo ele é

```
amt = 6 * 12.0;
PI = 3.14;
volLata = 0.335;
```

	Número	Tipo	Comentário
✓	6	int	Um inteiro não tem parte fracionária.
✓	-6	int	Inteiros podem ser negativos.
✓	0	int	Zero é um inteiro.
✓	0.5	double	Um número com parte fracionária é do tipo <code>double</code> .
✓	1.0	double	Um inteiro com parte fracionária <code>.0</code> é do tipo <code>double</code> .
✓	1E6	double	Um número em notação exponencial: $1 \times 10^6$ ou 1000000. Números em notação exponencial sempre são do tipo <code>double</code> .
✓	2.96E-2	double	Expoente negativo: $2.96 \times 10^{-2} = 2.96/100 = 0.0296$ .
✗	100,000		<b>ERRO:</b> não use vírgula como separador ou ponto decimal.
✗	3 1/2		<b>ERRO:</b> não use frações, use a notação decimal (3.5).

# Números em Ponto-Flutuante

- Java armazena números com partes fracionárias como números de ponto-flutuante
- Eles são armazenados em quatro partes
  - Sinal
  - Mantissa
  - Base
  - Expoente
- Por exemplo:  $-5 \times 10^0$
- Um `double` é um número de dupla precisão: ele ocupa 2 vezes o espaço de armazenamento (mantissa de 52 bits) do que o `float` (mantissa de 23 bits).

# Nomes de Variáveis

- Os nomes deveriam descrever o propósito da variável
- Use estas regras simples
  - 1 Nomes de variáveis devem iniciar com uma letra, com o caractere “sublinhado” (`_`) ou com o caractere “cifrão” (`$`), e continuar com letras (maiúsculas ou minúsculas), dígitos, sublinhado ou cifrao
  - 2 Você não pode usar outros símbolos (`?`, `%`, etc.), nem espaço.
  - 3 Separe palavras com a notação “camelHump”: comece com minúsculas e use letras maiúsculas para marcar a separação de palavras
  - 4 Não use palavras-reservadas de Java

# Nomes de Variáveis Legais e Ilegais em Java

	Nome de variável	Comentário
✓	volLata1	Nomes de variáveis consistem de letras, números e sublinhado.
✓	x	Em matemática, usa-se nomes curtos de variáveis, tal como $x$ ou $y$ . Eles são permitidos em Java, mas não são muito comuns, pois eles tornam o programa mais difícil de entender.
!	VolLata	<b>Atenção:</b> Java distingue maiúsculas de minúsculas. Este nome de variável é diferente de <code>volLata</code> , e viola a convenção de se iniciar nomes de variáveis com minúsculas.
✗	6latas	<b>Erro:</b> Nomes de variáveis não podem iniciar com números.
✗	vol lata	<b>Erro:</b> Nomes de variáveis não podem conter espaços.
✗	double	<b>Erro:</b> Você não pode usar palavras-reservadas como nome de variáveis.
✗	lit/pac.1	<b>Erro:</b> Você não pode usar símbolos como <code>/</code> ou <code>..</code> .



# Expressões de Atribuição

- Use a “expressão de atribuição” (com um =) para colocar um novo valor em uma variável

```
int latasPorPacote = 6; // declara e inicializa  
latasPorPacote = 12;   // atribui
```

- Cuidado: o sinal = **NÃO** é usado para comparações
  - Ele copia o valor calculado no lado direito da expressão para a variável que está à sua esquerda.
  - Aprenderemos sobre comparações mais tarde

# Sintaxe da Atribuição

- O valor calculado no lado direito do operador de atribuição (=) é copiado para a variável que está à sua esquerda

```
double total = 0;    /* Isto e' a inicializacao
                      de uma nova variavel e
                      nao uma atribuicao */

...
// Isto e' uma atribuicao
total = garrafas * VOLUME_GARRAFA;

...
// O mesmo nome pode aparecer nos dois lados de uma atribuicao
total = total + latas * VOLUME_LATA;
```

# Atualização de uma Variável

Passo-a-passo: `volumeTotal = volumeTotal + 2;`

- 1 Calcule o lado direito da atribuição. Encontre o valor de `volumeTotal`, e adicione 2 a ele
- 2 Armazene o resultado na variável indicada no lado esquerdo do operador de atribuição (neste caso, `totalVolume`)

# Constantes

- Quando uma variável é definida com a palavra-reservada `final`, o seu valor não poderá ser alterado

```
final double VOLUME_GARRAFA = 2;
```

- É uma boa prática usar constantes rotuladas para explicar valores a serem usados em cálculos
  - Qual forma é mais clara?

```
double volumeTotal = garrafas * 2;  
double volumeTotal = garrafas * VOLUME_GARRAFA;
```

- Um programador lendo a primeira sentença pode não entender o significado de 2
- Da mesma forma, se a constante for usada em vários lugares e necessitar ser alterada, bastará alterar a inicialização

# Declaração de Constantes

- A palavra-reservada `final` indica que o valor não poderá ser modificado
- por exemplo:

```
final double VOLUME_LATA = 0.355; // Litros em uma lata
```

- Não é obrigatório, mas costuma-se usar letras maiúsculas para constantes
- O comentário explica como o valor da constante foi definido

# Comentários em Java

- Lembre-se: há 3 tipos de comentários em Java

```
// Comentario de uma unica linha

/*
    Comentario de multiplas linhas
*/

/**
    Comentarios JavaDoc
    @author Aqui Vai o Seu Nome
    @version 15 ago. 2022
*/
```

- O compilador ignora comentários (comentários JavaDoc geram documentação)
- Identifique assunto, autoria e versão em um comentário JavaDoc em cada programa
- Use comentários para adicionar explicações para facilitar o entendimento do código

# Exemplo de Programa Comentado em Java

```
/**
    Este programa calcula o volume (em litros) de um pacote de
    6 latas de refrigerante e de uma garrafa de 2 litros de refrigerante
 */

public class Volume1 {
    public static void main(String[] args) {
        int latasPorPacote = 6;
        final double VOLUME_LATA = 0.355; // Litros em uma lata
        double volumeTotal = latasPorPacote * VOLUME_LATA;

        System.out.print("Um pacote de 6 latas de refrigerante contem ");
        System.out.print(volumeTotal);
        System.out.println(" litros.");
    }
}
```

# Erros Comuns

- Variáveis não declaradas

- Você deve declarar as variáveis antes de usá-las: (ou seja, acima no código)

```
double volumeLatas = 6 * litrosPorLata; // ??  
double litrosPorLata = 0.355;
```

- Variáveis não inicializadas

- Você deve inicializar as variáveis (isto é, definir um conteúdo para elas) antes de usá-las

```
int garrafas;  
int volumeGarrafas = garrafas * 2; // ??
```



# Erros Comuns

- *Overflow* significa que a área de armazenamento da variável não pode guardar o valor

```
int cinquentaMilhoes = 50000000;  
System.out.println(100 * cinquentaMilhoes);  
// Esperava-se 5000000000
```

- Será impresso 705032704
- Por quê?
  - O resultado (5 bilhões) estourou a capacidade de uma variável `int`
  - O valor máximo para um `int` é **+2,147,483,647**
- Use um `long` em vez de um `int` (ou um `double`)

# Erros Comuns

- Erros de arredondamento: valores de ponto-flutuante não são exatos (isto é uma limitação da conversão entre decimal e binário)

```
double preco = 4.35;  
double quantidade = 100;  
double total = preco * quantidade;  
// Deveria ser 100 * 4.35 = 435.00  
System.out.println(total); // Imprime 434.99999999999999
```

- Você pode lidar com erros deste tipo arredondando o valor para o inteiro mais próximo ou exibindo um número fixo de dígitos após o ponto decimal.

# Tipos Numéricos em Java

## • Tipos Inteiros

- `byte`: um número muito pequeno de 1 byte ou 8 bits (-128 to +127)
- `short`: um número pequeno (-32768 to +32767)
- `int`: um número grande (-2,147,483,648 ou `Integer.MIN_VALUE` até +2,147,483,647 ou `Integer.MAX_VALUE`)
- `long`: um número muito grande, com aproximadamente 19 dígitos decimais

## • Tipos de Ponto-Flutuante

- `float`: um número fracionário de precisão simples com cerca de 7 casas decimais e intervalo aproximado de  $\pm 10^{38}$
- `double`: um número fracionário de dupla precisão, para matemática pesada, com cerca de 15 casa decimais e intervalo aproximado de  $\pm 10^{308}$

## • Outros Tipos

- `boolean`: `true` (verdadeiro) ou `false` (falso)
- `char`: um símbolo ou caracter na codificação Unicode

# Armazenamento (em Bytes) por Tipo

- Tipos Inteiros

- byte: 1 byte
- short: 2 bytes
- int: 4 bytes
- long: 8 bytes

- Tipos de Ponto-Flutuante

- float: 4 bytes
- double: 8 bytes

- Outros Tipos

- boolean: 1 bit
- char: 2 bytes

# Aritmética

# Aritmética

- Java suporta as mesmas operações básicas que uma calculadora
  - parênteses
  - operadores unários (sinal):  $-a$ ,  $+a$
  - operadores binários: multiplicação ( $a * b$ ), divisão inteira ou real ( $a/b$ ), resto da divisão inteira ( $a \% b$ ), adição ( $a + b$ ), subtração ( $a - b$ )
- Porém, as expressões devem escritas de uma forma diferente:
  - Em Álgebra:

$$x = \frac{a + b}{2}$$

- Em Java: `x = (a + b) / 2.0;`
- A precedência nas expressões em Java é similar à precedência da Álgebra:
  - Parenteses, Operadores Unários, Multiplicação/Divisão/Resto, Soma/Subtração
  - Em caso de empate: resolver da esquerda para a direita

# Divisão Inteira e Resto da Divisão Inteira

- Quando ambas as partes de uma divisão são inteiras, o resultado será um inteiro
  - Toda informação fracionária será perdida e o resultado não será arredondado

```
int resultado = 7 / 4;
```

- O valor de `resultado` será 1
- Se você estiver interessado no resto da divisão de dois inteiros, use o operador `%` (chamado módulo)
  - Por exemplo, para:

```
int resto = 7 % 4;
```

- O valor de `resto` será 3
- O resto da divisão também é chamado de módulo da divisão

# Exemplos de Divisão Inteira e Resto da Divisão Inteira

Expressão (onde $n=1729$ )	Valor	Comentário
$n \% 10$	9	$n \% 10$ é sempre o último dígito de $n$
$n / 10$	172	Isto será sempre $n$ sem o último dígito
$n \% 100$	29	Os últimos 2 dígitos de $n$
$n / 10.0$	172.9	Como $10.0$ é um número em ponto-flutuante, a parte fracionária não é descartada
$-n \% 10$	-9	Como o primeiro número é negativo, o resto também é
$n \% 2$	1	$n \% 2$ é 0 se $n$ for par, 1 ou -1 se $n$ for ímpar

- Isto é muito útil para calcular troco

```
int totalCentavos = 1729;
int reais = totalCentavos / 100;    // 17
int centavos = totalCentavos % 100; // 29
```



# Potências e Raízes

- Em Java, não há símbolos para potência e raiz
- Por exemplo,

$$b \times \left(1 + \frac{r}{100}\right)^n$$

- Será implementado como: `b * Math.pow(1 + r / 100, n)`

$$b * \underbrace{\text{Math.pow}\left(1 + \frac{r}{100}, n\right)}$$

$$\underbrace{\frac{r}{100}}$$

$$1 + \frac{r}{100}$$

$$\underbrace{\left(1 + \frac{r}{100}\right)^n}$$

$$\underbrace{b \times \left(1 + \frac{r}{100}\right)^n}$$

- Em Java, a classe `Math` disponibiliza uma série de funções matemáticas

# Métodos Matemáticos

Método	Retorna
<code>Math.sqrt(x)</code>	Raiz quadrada de $x$ ( $\geq 0$ )
<code>Math.pow(x, y)</code>	$x^y$
<code>Math.sin(x)</code>	Seno de $x$ ( $x$ em radianos)
<code>Math.cos(x)</code>	Cosseno de $x$ ( $x$ em radianos)
<code>Math.tan(x)</code>	Tangente de $x$ ( $x$ em radianos)
<code>Math.toRadians(x)</code>	Converte $x$ graus para radianos (isto é, retorna $x \cdot \pi/180$ )
<code>Math.toDegrees(x)</code>	Converte $x$ radianos para graus (isto é, retorna $x \cdot 180/\pi$ )
<code>Math.exp(x)</code>	$e^x$
<code>Math.log(x)</code>	Logaritmo natural ( $\ln(x)$ , $x > 0$ )
<code>Math.log10(x)</code>	Logaritmo decimal ( $\log_{10}(x)$ , $x > 0$ )
<code>Math.round(x)</code>	Inteiro mais próximo de $x$ (como um <code>long</code> )
<code>Math.abs(x)</code>	Absolute value $ x $
<code>Math.max(x, y)</code>	O maior valor entre $x$ e $y$
<code>Math.min(x, y)</code>	O menor valor entre $x$ e $y$
...	...

# Misturando Tipos Numéricos

- É mais seguro converter um valor de um tipo inteiro para um tipo de ponto-flutuante, para não perder a “precisão”
- Fazer de outra forma (`double`  $\rightarrow$  `int`) pode ser perigoso
  - Toda informação fracionária é perdida
  - A parte fracionária é descartada (e não arredondada)
- Se você misturar os tipos inteiro e de ponto-flutuante em uma expressão, nenhuma precisão será perdida, pois o resultado será de ponto-flutuante:

```
double area, pi = 3.14;  
int raio = 3;  
area = raio * raio * pi;
```

# Conversão de Ponto-Flutuante para Inteiro

- O compilador Java não permite atribuição direta de valores em ponto flutuante para variáveis inteiras:

```
double balanço = total + taxa;  
int reais = balanço; // Erro!
```

- Você pode usar um conversor de tipo (cast) para forçar a conversão:

```
double balanço = total + taxa;  
int reais = (int)balanço; // Sem erro!
```

- Perde-se a parte fracionária do valor em ponto-flutuante (não é usado nenhum arredondamento)

## Sintaxe da Conversão Explícita de Tipo (*cast*)

- Um operador de conversão explícita corresponde ao nome de um tipo entre parênteses e é aplicado ao resultado de uma expressão
- Por exemplo, em `(int) (balanco*100)`
  - `balanco` é uma variável `double` e `100` é uma constante inteira
  - `balanco*100` resultará em um valor `double`
  - `(int)` faz com que o valor `double` seja convertido para `int` antes de ser usado
- A conversão explícita de tipo é uma ferramenta bastante poderosa e deve ser usada com cuidado
- Para arredondar um número de ponto-flutuante para o número inteiro mais próximo, use o método `Math.round`
- Este método retorna um inteiro do tipo `long`, porque números de ponto-flutuante maiores não podem ser armazenados em uma variável `int`

```
long valArredondado = Math.round(balanco);
```

# Expressões Aritméticas

Expressão Matemática	Expressão Java	Comentários
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	Os parênteses são necessários, pois $x + y / 2$ corresponde a $x + \frac{y}{2}$ .
$\frac{xy}{2}$	<code>x * y / 2</code>	Parênteses não são necessários, pois operadores da mesma precedência são avaliados da esquerda para a direita.
$\left(1 + \frac{r}{100}\right)^n$	<code>Math.pow(1 + r / 100, n)</code>	Use <code>Math.pow(x, n)</code> para computar $x^n$ .
$\sqrt{a^2 + b^2}$	<code>Math.sqrt(a * a + b * b)</code>	<code>a * a</code> é mais simples do que <code>Math.pow(a, 2)</code> .
$\frac{i + j + k}{3}$	<code>(i + j + k) / 3.0</code>	Se <code>i</code> , <code>j</code> e <code>k</code> são inteiros, usar um denominador igual a <code>3.0</code> força a divisão a ocorrer em ponto-flutuante.
$\pi$	<code>Math.PI</code>	<code>Math.PI</code> é uma constante declarada na classe <code>Math</code> .

# Erros Comuns

- Divisão inteira não intencional

```
System.out.print("Informe os ultimos 3 resultados: ");  
int s1 = in.nextInt();  
int s2 = in.nextInt();  
int s3 = in.nextInt();  
double media = (s1 + s2 + s3) / 3; // Error
```

- Por quê?

- Todos os cálculos do lado direito são feitos antes e como há apenas variáveis inteiras, o compilador usará divisão inteira
- Então o resultado (um `int`) é atribuído a um `double`
- Não haverá parte fracionária no resultado `int`, assim zero (.0) será atribuído à parte fracionária da variável `double`

# Erros Comuns

- Parênteses desbalanceados... Qual está correto?

```
d = -(b * b - 4 * a * c) / (2 * a) ;
```

ou

```
d = -(b * b - (4 * a * c) / 2 * a) ;
```

- A contagem de ( e de ) deve ser igual
- *“Tudo que se abre deve ser fechado...”*



# Exercícios (1)

- 1 Qual será o valor das variáveis `a`, `b` e `c`, após cada uma das seguintes declarações e atribuições em Java?

a `int a = 5, b = 2, c = 10;`  
`a = b;`  
`b = c;`

b `int a = 5, b = 2, c = 10;`  
`b = c;`  
`a = b;`

## Exercícios (2)

- 2 Qual será o valor das variáveis *a*, *b* e *c*, após cada uma das seguintes declarações e atribuições em Java?

a `int a = 1, b = 2, c = 3;`  
`a = b + 1;`  
`b = a - 1;`  
`c = c + 1;`

b `int a = 1, b = 2, c = 3;`  
`a = a + 1;`  
`b = b * a;`  
`c = c - b;`

## Exercícios (3)

- 3 Qual será o valor da variável `x`, que foi declarada com valor inicial 15, após cada uma das operações indicada na seguinte sequência de atribuições em Java?

```
int x = 15;  
x = x + 3;  
x = x - 6;  
x = x / 2;  
x = 3 * x;
```

## Exercícios (4)

- 4 Considere as seguintes declarações em Java:

```
int a = 5;  
int b = 2;  
int aux;
```

Qual a sequência de atribuições que deverá ser feita para trocar o valor das variáveis `a` e `b`? Se achar necessário, use a variável auxiliar `aux`.

- 5 Qual a sequência de operações necessárias para intercambiar os valores de 3 variáveis inteiras `a`, `b` e `c` de modo que `a` fique com o valor de `b`, `b` fique com o valor de `c` e `c` fique com o valor de `a`? Se achar necessário, use a variável auxiliar `aux`.

```
int a = 10, b = 20, c = 30, aux;
```

## Exercícios (5)

- 6 Converta cada uma das expressões algébricas a seguir para expressões válidas na linguagem Java. Considere que todas as variáveis são do tipo `double` e que têm valores válidos.

a

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

b

$$x = (a + b)^2 \cdot \frac{(2a + 5b)}{2} + 1$$

c

$$x = \frac{a^2 - 5b + \frac{c}{2}}{\frac{(b+3)}{7}}$$

d

$$x = \frac{(a + b^2)(a - b) + 4a - 5b + c}{2}$$

# Incrementando uma Variável

- Passo-a-passo: `contador = contador + 1;`
  - Faça o lado direito da atribuição antes: encontre o valor armazenado em `contador` e adicione 1 a ele
  - Armazene o resultado na variável que aparece no lado esquerdo do operador de atribuição (`contador` neste caso)
- Incrementar (+1) e decrementar (-1) tipos inteiros é tão comum que há versões abreviadas para cada forma longa

Forma Longa	Forma Abreviada
<code>contador = contador + 1;</code>	<code>contador++;</code>
<code>contador = contador - 1;</code>	<code>contador--;</code>

## Usando Pré- e Pós-incremento/decremento

- Usar, por exemplo, `i++`; é equivalente a usar `++i`;
- O mesmo vale para `i--`; e `--i`;
- Mas os pré- e pós-incrementos/decrementos podem aparecer em expressões, neste caso haverá diferença
- Por exemplo:
  - `x = i++`; corresponde a `x = i`; `i = i + 1`;
  - `x = --i + j`; corresponde a `i = i - 1`; `x = i + j`;
- Se `++` ou `--` aparecerem antes do nome da variável, ela será incrementada ou decrementada antes de seu valor ser usado
- Se aparecerem depois, ela será incrementada ou decrementada somente depois de seu valor ser usado

# Atribuições Sintéticas (1)

- Muitas vezes, quando a variável que recebe o resultado de uma atribuição (lado esquerdo da atribuição) aparece também na expressão (lado direito), é possível escrever a atribuição de uma forma mais curta
- Em vez de escrever:  
`<var> = <var> <op> <expressão>;`  
pode-se escrever:  
`<var> <op>= <expressão>;`



## Atribuições Sintéticas (2)

- Por exemplo, as linhas:

```
a = a + 10;  
x = x * (b/c) ;  
n = n - z;  
w = w / (p + q) ;
```

- Poderiam ser escritas como:

```
a += 10;  
x *= (b/c) ;  
n -= z;  
w /= (p + q) ;
```

# Exercício

Reescreva da forma mais abreviada possível os seguintes trechos de código em Java:

a) `i = i - 1;`

b) `b = b + 1;`  
`a = b * c;`  
`c = c - 1;`

c) `x = x / (b + c);`

d) `x = x - 1;`  
`z = z - x + y;`  
`y = y + 1;`

# Exercício (Solução)

a) `--i;        // ou:    i--;`

b) `a = ++b * c--;`

c) `x /= (b + c);        // ou:    x /= b + c;`

d) `z -= --x - y++;        // ou:    z += -(--x) + y++;`

# Entrada e Saída

# Lendo a Entrada

- Você pode ter que pedir por uma entrada do usuário
- Esta entrada é obtida do teclado
- Por enquanto, não se preocupe com os detalhes, apenas siga os seguintes passos:

1 Importe a classe `Scanner` do pacote `java.util`

```
import java.util.Scanner;
```

2 Crie um objeto da classe `Scanner`:

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

3 Use os métodos da classe `Scanner` para obter valores de entrada:

```
int garrafas = in.nextInt();  
double preco = in.nextDouble();
```

- Classes Java são agrupadas em pacotes e o comando `import` permite usar classes de pacotes
- A classe `Scanner` permite que você leia a entrada do usuário a partir do teclado - ela faz parte do pacote `util` da API Java

# Exemplo de Leitura

```
// Esta linha permite usar a classe Scanner
import java.util.Scanner;
...
// Cria um objeto do tipo Scanner, chamado in,
// para ler a entrada do teclado
Scanner in = new Scanner(System.in);
...
// Mostre uma mensagem antes de obter a entrada com print()
System.out.print("Informe o numero de garrafas: ");
// Define uma variavel chamada garrafas e chama o metodo
// nextInt() do objeto in, que espera pela entrada do usuario
int garrafas = in.nextInt();
```

# Exercício

- 1 Escreva um programa em Java que leia o raio de uma esfera, calculando e mostrando o valor de seu volume.

# Saída Formatada

- A saída de números em ponto-flutuante pode parecer estranha

Preco por litro: 1.215962441314554

- Para controlar a aparência da saída de variáveis numéricas usa-se ferramentas controle da saída, tais como:

```
System.out.print("Preco por litro: ");  
System.out.printf("%.2f", price);
```

- O resultado será:

Preco por litro: 1,22

- Observe que o separador para casas decimais foi adequado para as definições de país e língua locais.



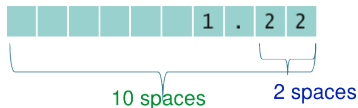
# Saída Formatada

- Também pode-se usar:

```
System.out.print("Preço por litro: ");  
System.out.printf("%10.2f", price);
```

- Que gera:

Preço por litro: 1,22



- `%.2f` e `%10.2f` são especificadores de formato

# Tipos de Formatação

- A formatação é bastante útil para alinhar colunas na saída

Código	Tipo	Exemplo
d	Inteiros decimais	123
f	Ponto-flutuante fixo	12.30
e	Ponto-flutuante exponencial	1.23e+1
g	Ponto-flutuante genérico (notação exponencial será usada para números muito grandes ou muito pequenos)	12.3
s	Cadeias de caracteres ( <i>strings</i> )	Tax:

- Você também pode incluir texto dentro das aspas:

```
System.out.printf("Preco por litro: %10.2f", price);
```

- Ou incluir mais de um especificador de formato dentro das aspas e quebras de linha (\):

```
System.out.printf("As raizes sao: %10.4f e %10.4f\n", x1,x2);
```

# Modificadores de Formatação

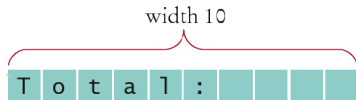
- Você também pode usar modificadores de formatação para alterar o modo como valores numéricos e textos são exibidos

Modificador	Significado	Exemplo
-	Alinhamento à esquerda	1.23 (seguido por espaços)
0	Preenchimento com zeros	001.23
+	Mostra o sinal também para números positivos	+1.23

# Exemplos de Modificadores de Formatação

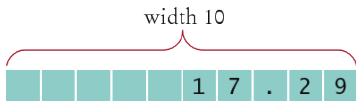
- Alinhamento à esquerda de um *string*:

```
System.out.printf("%-10s", "Total:");
```



- Alinhamento à direita de um número com 2 casas decimais:

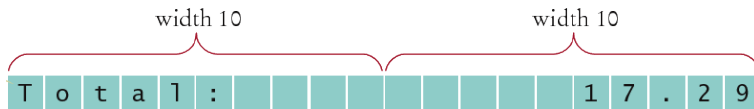
```
System.out.printf("%10.2f", price);
```



## Exemplos de Modificadores de Formatação

- Impressão de múltiplos valores:

```
System.out.printf("%-10s%10.2f", "Total:", price);
```



# Volume2.java (HORSTMANN, 2013, p. 51-52)

```
import java.util.Scanner;

/**
 * This program prints the price per ounce for a six-pack of cans.
 */
public class Volume2 {
    public static void main(String[] args) {
        // Read price per pack
        Scanner in = new Scanner(System.in);
        System.out.print("Please enter the price for a six-pack: ");
        double packPrice = in.nextDouble();

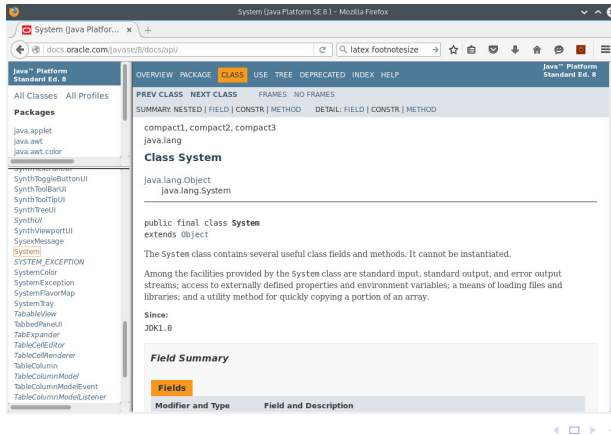
        // Read can volume
        System.out.print("Please enter the volume for each can (in ounces): ");
        double canVolume = in.nextDouble();

        // Compute pack volume
        final double CANS_PER_PACK = 6;
        double packVolume = canVolume * CANS_PER_PACK;

        // Compute and print price per ounce
        double pricePerOunce = packPrice / packVolume;
        System.out.printf("Price per ounce: %8.2f", pricePerOunce);
        System.out.println();
    }
}
```

# Documentação da API Java

- A lista de classes e métodos da API Java pode ser obtida em:  
<http://docs.oracle.com/javase/8/docs/api/>

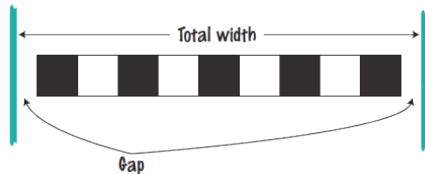


# Resolução de Problemas: Primeiro Faça à Mão



# Resolução de Problemas: Primeiro Faça à Mão

- Um passo bastante importante para desenvolver um algoritmo é primeiro realizar as computações à mão.
- Por exemplo:
  - Uma linha de azulejos pretos e brancos deve ser colocada ao longo de uma parede. Por razões estéticas, o arquiteto especificou que o primeiro e o último bloco devem ser pretos.
  - Sua tarefa é computar o número de azulejos necessários e o espaço de sobra em cada extremidade, dado o espaço disponível e o comprimento de cada azulejo.



# Inicie com exemplos de teste

- Valores iniciais de teste
  - Espaço total:  $1000cm$
  - Tamanho de cada azulejo:  $50cm$
- Teste seus valores
  - Vejamos...  $1000cm/50cm = 20$ , perfeito! 20 azulejos. Nenhuma sobra.
  - Mas, espere... Preto, Branco; Preto, Branco; ... "O primeiro e o último azulejo devem ser pretos."
- Olhe mais atentamente para o problema...
  - Inicie com um azulejo preto, então some determinado número de pares de azulejos branco-preto



- Observação: cada par tem 2 vezes o tamanho de um azulejo (em nosso exemplo:  $2 \times 50cm = 100cm$ )

# Continue Trabalhando na sua Solução

- Calcule o comprimento total dos azulejos
  - Um azulejo preto:  $50cm$
  - 9 pares de azulejos branco-preto:  $900cm$
  - Comprimento total:  $950cm$
- Cálculo da sobra (uma em cada extremidade):
  - $1000cm - 950cm = 50cm$  de sobra total
  - $50cm/2 = 25cm$  em cada extremidade

# Agora Pense no Algoritmo

- Use o exemplo para ver como os valores foram calculados

- Quantos pares? (deve ser um número inteiro)

$$\text{numPares} = (\text{int}) (\text{compTotal} - \text{compAzulejo}) / (2 \times \text{compAzulejo})$$

- Quantos azulejos?

$$\text{numAzulejos} = 1 + 2 \times \text{numPares}$$

- Sobra em cada extremidade?

$$\text{sobra} = (\text{compTotal} - \text{numAzulejos} \times \text{compAzulejo}) / 2$$

# Exercício

Considere um terreno retangular sobre o qual se deseja colocar “placas” de grama que são vendidas por metro quadrado. Neste terreno há um silo redondo e uma casa em formato retangular. Determine os dados que devem ser lidos para que se saiba o custo da colocação de grama nesse terreno e faça o programa em Java para realizar este cálculo.



# Cadeias de Caracteres (Strings)

# Strings

- O tipo `String` é usado para armazenamento de textos (sequências de caracteres)
- Sintaxe:

```
// <tipo> <nome variavel> = <conteudo inicial> ;  
String nome = "Texto";
```

- Uma vez que você tenha uma variável do tipo `String`, você pode usar métodos como:

```
int n = nome.length(); // n recebe o valor 5
```

- O comprimento de um *string* corresponde ao número de caracteres que ele contém
  - Um *string* vazio (tamanho 0) corresponde a ""
  - O tamanho máximo de um *string* é bastante grande (um `int`)

# Concatenação de *Strings* (+)

- Você pode “adicionar” um *string* no final de outro:

```
String pNome = "Joao";  
String uNome = "Silva";  
String nome = pNome + uNome; // JoaoSilva
```

- Quer um espaço entre os nomes?

```
String nome = pNome + " " + uNome; // Joao Silva
```

- Para concatenar uma variável numérica a um *string*:

```
String a = "Agente";  
int n = 7;  
String bond = a + n; // Agente7
```

- Concatenar *strings* e números em `println`:

```
System.out.println("Total = " + total);
```



# Entrada de *Strings*

- Você pode ler um *string* do terminal com `next`
  - O método `next` lê uma palavra de cada vez
  - Palavras lidas por `next` são separadas por espaços

```
System.out.print("Digite seu primeiro nome: ");  
String nome = in.next();
```

- Você pode ler uma linha inteira do terminal com `nextLine`
  - O método `nextLine` faz a leitura até encontrar um “Enter”

```
System.out.print("Forneca o seu endereço: ");  
String endereco = in.nextLine();
```

- Para converter uma variável `String` para um número:

```
System.out.print("Digite a sua idade: ");  
String entrada = in.nextLine();  
int idade = Integer.parseInt(entrada); // apenas digitos
```

# Sequências de Escape em *Strings*

- Como imprimir aspas duplas?
  - Coloque uma \ antes do ", dentro do *string*

```
System.out.print("Ele disse \"Alo\"");
```

- Ok, e agora como imprimir uma contra-barra?
  - Coloque uma \ antes da \!

```
System.out.print("C:\\Temp\\Secret.txt");
```

- Mude de linha dentro de um *string* (\n):

```
System.out.print("*\n**\n***\n");
```

# Strings e Caracteres

- *Strings* são sequências de caracteres
  - Caracteres *Unicode*, para ser mais exato
  - Caracteres tem seu próprio tipo: `char`
  - Caracteres têm valores numéricos
    - Consulte uma tabela ASCII para descobrir o valor de cada código
    - Por exemplo, a letra 'A' teria o valor 65 se fosse um número
- Usa-se apóstrofos ao redor de caracteres:

```
char initial = 'B';
```

- Usa-se aspas ao redor de *strings*:

```
String initials = "BRL";
```

# Obtendo um caracter de um *String*

- Cada caracter dentro de um *string* tem um índice numérico
- O primeiro caracter tem o índice 0
- O método `charAt` retorna o caracter com determinado índice em um *string*

```
String nome = "Harry";  
char inicio = nome.charAt(0);  
char fim = nome.charAt(4);
```

0	1	2	3	4
H	a	r	r	y

# Obtendo uma Parte de um *String*

- Um *substring* é uma parte de um *string*
- O método `substring(índice inicial, limite final)` retorna um conjunto de caracteres a partir de uma posição (índice inicial) de um *string*

```
String greeting = "Hello!";  
String sub = greeting.substring(0, 2); // = "He"  
String sub2 = greeting.substring(3, 5); // = "lo"
```

# Obtendo uma Parte de um *String*

```
public class TesteDeSubstring {
    public static void main (String args[])
    {
        String texto = "01234";
        System.out.println("["+texto.substring(0)+"]"); // [01234]
        System.out.println("["+texto.substring(1)+"]"); // [1234]
        System.out.println("["+texto.substring(2)+"]"); // [234]
        System.out.println("["+texto.substring(3)+"]"); // [34]
        System.out.println("["+texto.substring(4)+"]"); // [4]
        System.out.println("["+texto.substring(5)+"]\n"); // []
        /* ERRO DE EXECUCAO:
        System.out.println("["+texto.substring(1,0)+"]"); */
        System.out.println("["+texto.substring(1,1)+"]"); // []
        System.out.println("["+texto.substring(1,2)+"]"); // [1]
        System.out.println("["+texto.substring(1,3)+"]"); // [12]
        System.out.println("["+texto.substring(1,4)+"]"); // [123]
        System.out.println("["+texto.substring(1,5)+"]"); // [1234]
    }
}
```

# Operações sobre *Strings*

Statement	Result	Comment
<code>string str = "Ja"; str = str + "va";</code>	str is set to "Java"	When applied to strings, + denotes concatenation.
<code>System.out.println("Please" + " enter your name: ");</code>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<code>team = 49 + "ers"</code>	team is set to "49ers"	Because "ers" is a string, 49 is converted to a string.
<code>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</code>	first contains "Harry" last contains "Morgan"	The next method places the next word into the string variable.
<code>String greeting = "H &amp; S"; int n = greeting.length();</code>	n is set to 5	Each space counts as one character.
<code>String str = "Sally"; char ch = str.charAt(1);</code>	ch is set to 'a'	This is a char value, not a String. Note that the initial position is 0.
<code>String str = "Sally"; String str2 = str.substring(1, 4);</code>	str2 is set to "all"	Extracts the substring starting at position 1 and ending before position 4.
<code>String str = "Sally"; String str2 = str.substring(1);</code>	str2 is set to "ally"	If you omit the end position, all characters from the position until the end of the string are included.
<code>String str = "Sally"; String str2 = str.substring(1, 2);</code>	str2 is set to "a"	Extracts a String of length 1; contrast with <code>str.charAt(1)</code> .
<code>String last = str.substring( str.length() - 1);</code>	last is set to the string containing the last character in str	The last character has position <code>str.length() - 1</code> .

# Principais métodos de *String* (1)

- `char charAt(int index)`: retorna o valor do caractere no índice especificado
- `int compareTo(String anotherString)`: comparação lexicográfica (diferenciando caixa alta e caixa baixa)
- `int compareToIgnoreCase(String str)`: comparação lexicográfica (ignorado caixa alta e caixa baixa)
- `String concat(String str)`: concatena a *string* especificada no final desta *string*
- `boolean endsWith(String suffix)`: verifica se a *string* termina com o sufixo especificado
- `boolean equals(Object anObject)`: verifica se a *string* especificada é igual a esta
- `boolean equalsIgnoreCase(String anotherString)`: verifica se a *string* especificada é igual a esta, sem diferenciar caixa alta e caixa baixa



## Principais métodos de *String* (2)

- `int hashCode()`: retorna um *hash code* para esta *string*
- `boolean isEmpty()`: retorna `true` se, e somente se, o `length()` for igual a 0
- `int length()`: retorna o tamanho desta *string*
- `String substring(int beginIndex)`: retorna uma nova *string* que é uma *substring* desta *string*, do índice até o final
- `String substring(int beginIndex, int endIndex)`: retorna uma nova *string* que é uma *substring* desta *string*

## Principais métodos de *String* (3)

- `String toLowerCase()`: converte todos os caracteres desta *string* para uma *string* em caixa baixa usando as regras da localidade padrão
- `String toLowerCase(Locale locale)`: converte todos os caracteres desta *string* para uma *string* em caixa baixa usando as regras da localidade especificada
- `String toString()`: este objeto (que já é uma *string*) é autoretornado
- `String toUpperCase()`: converte todos os caracteres desta *string* para uma *string* em caixa alta usando as regras da localidade padrão
- `String toUpperCase(Locale locale)`: converte todos os caracteres desta *string* para uma *string* em caixa alta usando as regras da localidade especificada

## Principais métodos de *String* (4)

- `String trim()`: retorna uma cópia da *string* eliminando espaços iniciais e finais
- `static String valueOf(boolean b)`: retorna uma *string* representando o argumento `boolean`
- `static String valueOf(char c)`: retorna uma *string* representando o argumento `char`
- `static String valueOf(double d)`: retorna uma *string* representando o argumento `double`
- `static String valueOf(float f)`: retorna uma *string* representando o argumento `float`
- `static String valueOf(int i)`: retorna uma *string* representando o argumento `int`
- `static String valueOf(long l)`: retorna uma *string* representando o argumento `long`
- ...

## Referências

# Referências

HORSTMANN, C. **Java for Everyone – Late Objectt.** 2. ed. Hoboken: Wiley, 2013. xxxiv, 589 p.