

A large, light blue, 3D-style number '5' is centered in the background. The text 'Tipos, literais, operadores e controle de fluxo' is overlaid on this number in a bold, dark blue font with a white drop shadow.

Tipos, literais, operadores e controle de fluxo

Helder da Rocha
www.argonavis.com.br

Operadores e controle de fluxo da execução

- *Este módulo explora as estruturas procedurais da linguagem Java*
- *Operadores*
 - *Aritméticos, lógicos, binários, booleanos, de deslocamento, de concatenação, de conversão, ...*
- *Conversão de tipos*
 - *Promoção*
 - *Coerção (cast)*
- *Estruturas de controle de execução*
 - *if-else,*
 - *for, while, do-while*
 - *break, continue, rótulos*
 - *switch (case)*

- *Um operador produz um novo valor a partir de um ou mais argumentos*
- *Os operadores em Java são praticamente os mesmos encontrados em outras linguagens*
 - *+, -, /, *, =, ==, <, >, >=, &&, etc.*
- *A maior parte dos operadores só trabalha com valores de tipos primitivos.*
- *Exceções:*
 - *+ e += são usados na concatenação de strings*
 - *!=, = e == são usados também com objetos (embora não funcionem da mesma forma quanto aos valores armazenados nos objetos)*

Lista de operadores do Java

OPERADOR	FUNÇÃO	OPERADOR	FUNÇÃO
+	Adição	~	Complemento
-	Subtração	<<	Deslocamento à esquerda
*	Multiplicação	>>	Deslocamento à direita
/	Divisão	>>>	Desloc. a direita com zeros
%	Resto	=	Atribuição
++	Incremento	+=	Atribuição com adição
--	Decremento	-=	Atribuição com subtração
>	Maior que	*=	Atribuição com multiplicação
>=	Maior ou igual	/=	Atribuição com divisão
<	Menor que	%=	Atribuição com resto
<=	Menor ou igual	&=	Atribuição com AND
==	Igual	=	Atribuição com OR
!=	Não igual	^=	Atribuição com XOR
!	NÃO lógico	<<=	Atribuição com desl. esquerdo
&&	E lógico	>>=	Atribuição com desl. direito
	OU lógico	>>>=	Atrib. C/ desl. a dir. c/ zeros
&	AND	? :	Operador ternário
^	XOR	(tipo)	Conversão de tipos (cast)
	OR	instanceof	Comparação de tipos

Precedência

- A **precedência** determina em que ordem as operações em uma expressão serão realizadas.

- Por exemplo, operações de multiplicação são realizadas antes de operações de soma:

```
int x = 2 + 2 * 3 - 9 / 3; // 2+6-3 = 5
```

- **Parênteses** podem ser usados para sobrepor a precedência

```
int x = (2 + 2) * (3 - 9) / 3; // 4*(-6)/3 = -8
```

- A maior parte das expressões de mesma precedência é calculada da **esquerda para a direita**

```
int y = 13 + 2 + 4 + 6; // (((13 + 2) + 4) + 6)
```

- Há exceções. Por exemplo, atribuição.

Tabela de precedência

ASSOC	TIPO DE OPERADOR	OPERADOR
D a E	separadores	[] . ; , ()
E a D	operadores unários	new (cast) +expr -expr ~ !
E a D	incr/decr pré-fixado	++expr --expr
E a D	multiplicativo	* / %
E a D	aditivo	+ -
E a D	deslocamento	<< >> >>>
E a D	relacional	< > >= <= instanceof
E a D	igualdade	== !=
E a D	AND	&
E a D	XOR	^
E a D	OR	
E a D	E lógico	&&
E a D	OU lógico	
D a E	condicional	?:
D a E	atribuição	= += -= *= /= %= >>= <<= >>>= &= ^= !=
E a D	incr/decr pós fixado	expr++ expr--

Literais de caracteres em Java

SEQÜÊNCIA	VALOR DO CARACTERE
<code>\b</code>	Retrocesso (backspace)
<code>\t</code>	Tabulação
<code>\n</code>	Nova Linha (new line)
<code>\f</code>	Alimentação de Formulário (form feed)
<code>\r</code>	Retorno de Carro (carriage return)
<code>\"</code>	Aspas
<code>\'</code>	Aspa
<code>\\</code>	Contra Barra
<code>\nnn</code>	O caractere correspondente ao valor octal <i>nnn</i> , onde <i>nnn</i> é um valor entre 000 e 0377.
<code>\unnnn</code>	O caractere Unicode <i>nnnn</i> , onde <i>nnnn</i> é de um a quatro dígitos hexadecimais. Seqüências Unicode são processadas antes das demais seqüências.

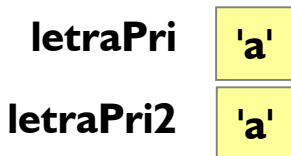
- A atribuição é realizada com o operador '='
 - '=' serve apenas para atribuição – não pode ser usado em comparações (que usa '==')!
 - Copia o valor da variável ou constante do lado direito para a variável do lado esquerdo.

```
x = 13; // copia a constante inteira 13 para x
y = x;  // copia o valor contido em x para y
```
- A atribuição copia **valores**
 - O valor armazenado em uma variável de tipo primitivo é o **valor** do número, caractere ou literal booleana (true ou false)
 - O valor armazenado em uma variável de tipo de classe (referência para objeto) é o **ponteiro** para o objeto ou null.
 - Conseqüentemente, copiar referências por atribuição **não copia objetos** mas apenas cria novas referências para o mesmo objeto!

Passagem de valores via atribuição

■ Variáveis de tipos primitivos

Pilha após linha 2



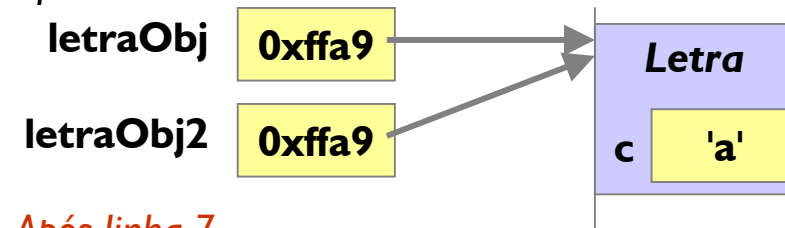
Pilha após linha 3



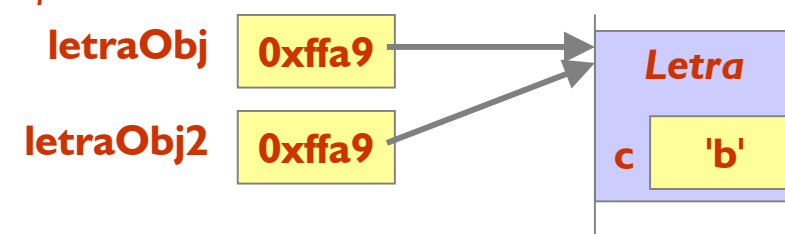
```
(...)  
1: char letraPri = 'a';  
2: char letraPri2 = letraPri;  
3: letraPri = 'b';  
(...)
```

■ Referências de objetos

Após linha 6



Após linha 7



```
public class Letra {  
    public char c;  
}
```

```
(...)  
4: Letra letraObj = new Letra();  
5: letraObj.c = 'a';  
6: Letra letraObj2 = letraObj;  
7: letraObj2.c = 'b';  
(...)
```

Operadores matemáticos

- **+** *adição*
- **-** *subtração*
- ***** *multiplicação*
- **/** *divisão*
- **%** *módulo (resto)*

- *Operadores unários*
 - **-n** e **+n** (ex: -23) (em uma expressão: $13 + -12$)
 - Melhor usar parênteses: $13 + (-12)$
- *Atribuição com operação*
 - **+=**, **-=**, ***=**, **/=**, **%=**
 - **x = x + 1** equivale a **x += 1**

Incremento e decremento

- *Exemplo*

```
int a = 10;
```

```
int b = 5;
```

- *Incrementa ou decrementa antes de usar a variável*

```
int x = ++a; // a contém 11, x contém 11
```

```
int y = --b; // b contém 4, y contém 4
```

- *A atribuição foi feita DEPOIS!*

- *Incrementa ou decrementa depois de usar a variável*

```
int x = a++; // a contém 11, x contém 10
```

```
int y = b--; // b contém 4, y contém 5
```

- *A atribuição foi feita ANTES!*

Operadores relacionais

- `==` *igual*
 - `!=` *diferente*
 - `<` *menor*
 - `<=` *menor ou igual*
 - `>` *maior*
 - `>=` *maior ou igual*
-
- *Sempre produzem um resultado booleano*
 - *true ou false*
 - *Comparam os valores de duas variáveis ou de uma variável e uma constante*
 - *Comparam as referências de objetos (apenas `==` e `!=`)*

Operadores lógicos

- **&&** *E (and)*
- **||** *Ou (or)*
- **!** *Negação (not)*

- *Produzem sempre um valor booleano*
 - *true ou false*
 - *Argumentos precisam ser valores booleanos ou expressões com resultado booleano*
 - *Por exemplo: (3 > x) && !(y <= 10)*
- *Expressão será realizada até que o resultado possa ser determinado de forma não ambígua*
 - *“short-circuit”*
 - *Exemplo: (false && <qualquer coisa>)*
 - *A expressão <qualquer coisa> não será calculada*

Operadores orientados a bit

- $\&$ *and*
- $|$ *or*
- \wedge *xor (ou exclusivo)*
- \sim *not*

- *Para operações em baixo nível (bit por bit)*
 - *Operam com inteiros e resultados são números inteiros*
 - *Se argumentos forem booleanos, resultado será igual ao obtido com operadores booleanos, mas sem 'curto-circuito'*
 - *Suportam atribuição conjunta: $\&=$, $|=$ ou $\wedge=$*

Operadores de deslocamento

- `<<` deslocamento de bit à esquerda
(multiplicação por dois)
- `>>` deslocamento de bit à direita
(divisão truncada por dois)
- `>>>` deslocamento à direita sem
considerar sinal (acrescenta zeros)
- Para operações em baixo nível (bit a bit)
 - Operam sobre inteiros e inteiros longos
 - Tipos menores (short e byte) são convertidos a int antes de realizar operação
 - Podem ser combinados com atribuição: `<<=`, `>>=` ou `>>>=`

Operador ternário (if-else)

- *Retorna um valor ou outro dependendo do resultado de uma expressão booleana*
 - `variavel = expressão ? valor, se true`
`: valor, se false;`
- *Exemplo:*

```
int x = (y != 0) ? 50 : 500;
String tit = (sex == 'f') ? "Sra." : "Sr"
num + " pagina" + (num != 1) ? "s" : ""
```
- *Use com cuidado*
 - *Pode levar a código difícil de entender*

Operador de concatenação

- *Em uma operação usando "+" com dois operandos, se um deles for String, o outro será convertido para String e ambos serão concatenados*
- *A operação de concatenação, assim como a de adição, ocorre da direita para a esquerda*

```
String s = 1 + 2 + 3 + "=" + 4 + 5 + 6;
```


- *Resultado: s contém a String "6=456"*

- *instanceof* é um operador usado para comparar uma referência com uma classe
 - A expressão será *true* se a referência for do tipo de uma classe ou subclasse testada e *false*, caso contrário
 - Sintaxe: referência instanceof Classe
- Exemplo:

```
if (obj instanceof Point) {  
    System.out.println("Descendente de Point");  
}
```

Tipos de dados

 *boolean (8 bits)*

 *byte (8 bits)*

 *char (16 bits)*

 *short (16 bits)*

 *int (32 bits)* *long (64 bits)*

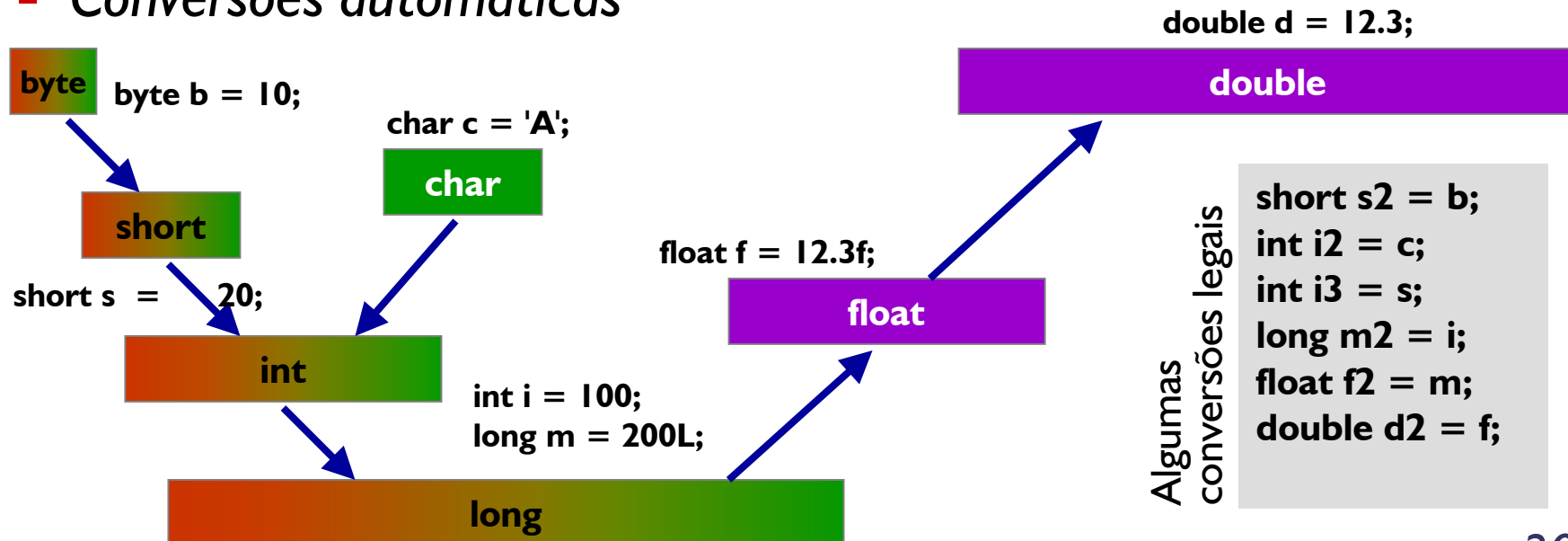


 *float (32 bits)* *double (64 bits)*



Conversão de tipos primitivos

- *Java converterá um tipo de dados em outro sempre que isto for apropriado*
- *As conversões ocorrerão automaticamente quando houver garantia de não haver perda de informação*
 - *Tipos menores em tipos maiores*
 - *Tipos de menor precisão em tipos de maior precisão*
 - *Tnteiros em ponto-flutuante*
- *Conversões automáticas*



Conversão de referências

- *Pode-se atribuir uma referência A a uma outra referência B de um tipo diferente, desde que*
 - *B seja uma **superclasse** (direta ou indireta) de A: Qualquer referência pode ser atribuída a uma referência da classe Object*
 - *B seja uma **interface** implementada por A: mais detalhes sobre interfaces em aulas futuras*


```
class Carro extends Veiculo {...}
class Veiculo implements Dirigivel {}
class Porsche extends Carro {...}
```

Algumas conversões legais

```
Carro c = new Carro();
Veiculo v = new Carro();
Object o = new Carro();
Dirigivel d = new Carro();
Carro p = new Porsche();
```

Mais detalhes sobre este assunto em capítulos posteriores!

Operadores de coerção

- Na coerção (*cast*), o **programador** assume os riscos da conversão de dados
 - No tipo *byte* cabem inteiros até 127
 - No tipo *short* cabem inteiros até 32767
 - Não há risco de perda de informação na atribuição a seguir
short s = 100; byte b = s;
(pois 100 cabe em *byte*) mas o compilador acusará erro porque um *short* não pode ser atribuído a *byte*.
 - Solução
byte b = (byte) s;
 operador de coerção (*cast*)
 - O programador "assume o risco", declarando entre parênteses, que o conteúdo de **s** cabe em **byte**.
 - O operador de coerção tem maior precedência que os outros operadores!

- Qualquer operação com dois ou mais operandos de tipos diferentes sofrerá **promoção**, isto é, conversão automática ao tipo mais abrangente, que pode ser
 - O maior ou mais preciso tipo da expressão (até double)
 - O tipo **int** (para tipos menores que int)
 - O tipo **String** (no caso de concatenações) - (na verdade **isto não é uma promoção**)

Exemplos

- `String s = 13 - 9 * 16 + "4" + 9 + 2; // "-131492"`
- `double d = 12 + 9L + 12.3; // tudo é promovido p/ double`
- `byte b = 9; byte c = 10; byte d = 12;`
`byte x = (byte) (b + c + d);`

promovidos para int!

a partir daqui só o sinal '+' é permitido!

*cast é essencial aqui!
Observe os parênteses!*

Controle de execução

- O controle do fluxo da execução em Java utiliza os mesmos comandos existentes em outras linguagens
 - Repetição: *for*, *while*, *do-while*
 - Seleção: *if-else*, *switch-case*
 - Desvios (somente em estruturas de repetição): *continue*, *break*, rótulos
- Não existe comando *goto*
 - *goto*, porém, é palavra-reservada.

- *Todas as expressões condicionais usadas nas estruturas for, if-else, while e do-while são expressões booleanas*
 - *O resultado das expressões deve ser sempre true ou false*
 - *Não há conversões automáticas envolvendo booleanos em Java (evita erros de programação comuns em C/C++)*

Código errado.
Não compila
em Java

```
int x = 10;  
if (x = 5) {  
    ...  
}
```

*código aceito em C/C++
(mas provavelmente errado)
x, com valor 5, converte-se
em 'true'.*

Código correto.
x == 5 é expressão
com resultado true
ou false

```
int x = 10;  
if (x == 5) {  
    ...  
}
```

■ Sintaxe

```
if (expressão booleana)  
    instrução_simples;
```

```
if (expressão booleana) {  
    instruções  
}
```

```
if (expressão booleana) {  
    instruções  
} else if (expressão booleana) {  
    instruções  
} else {  
    instruções  
}
```

■ Exemplo

```
if ( ano < 0 ) {  
    System.out.println("Não é um ano!");  
} else if ( (ano%4==0 && ano%100!=0) || (ano%400==0) ) {  
    System.out.println("É bissexto!");  
} else {  
    System.out.println("Não é bissexto!");  
}
```

- A palavra-chave *return* tem duas finalidades
 - Especifica o que um método irá retornar (se o método não tiver sido declarado com tipo de retorno void)
 - Causa o retorno imediato à linha de controle imediatamente posterior à chamada do método
- Exemplos de sintaxe:

```
boolean método() {  
    if (condição) {  
        instrução;  
        return true;  
    }  
    resto do método  
    return false;  
}
```

```
void método() {  
    if (condição) {  
        instrução;  
        return;  
    }  
    mais coisas...  
}
```

Este exemplo funciona como um if com else:

while e do-while

■ Sintaxe

```
while (expressão booleana )  
{  
    instruções;  
}
```

```
do  
{  
    instruções;  
} while (expressão booleana ) ;
```

■ Exemplos

```
int x = 0;  
while (x < 10) {  
    System.out.println ("item " + x);  
    x++;  
}
```

```
int x = 0;  
do {  
    System.out.println ("item " + x);  
    x++;  
} while (x < 10);
```

```
while ( true ) {  
    if (obj.z == 0) {  
        break;  
    }  
}
```

loop infinito!

■ Sintaxe

```
for ( inicialização ;
      expressões booleanas ;
      passo da repetição )
{
    instruções ;
}
```

```
for ( inicialização ;
      expressões booleanas ;
      passo da repetição )
    instrução_simples ;
```

■ Exemplos

```
for ( int x = 0; x < 10; x++ ) {
    System.out.println ("item " + x);
}
```

```
for ( int x = 0, int y = 25;
      x < 10 && (y % 2 == 0);
      x++, y = y - 1 ) {
    System.out.println (x + y);
}
```

```
for ( ; ; ) {
    if (obj.z == 0) {
        break;
    }
}
```

loop infinito!

break e continue

- **break**: interrompe a execução do bloco de repetição.
 - Continua com a próxima instrução, logo após o bloco.
- **continue**: interrompe a execução da iteração
 - Testa a condição e reinicia o bloco com a próxima iteração.

```
while (!terminado) {  
    passePagina();  
    if (alguemChamou == true) {  
        break;           // caia fora deste loop  
    }  
    if (paginaDePropaganda == true) {  
        continue;       // pule esta iteração  
    }  
    leia();  
}  
restoDoPrograma();
```

break e continue com rótulos

- **break** e **continue** sempre atuam sobre o bloco de repetição onde são chamados
- Em blocos de repetição contidos em outros blocos, pode-se usar **rótulos** para fazer break e continue atuarem em blocos externos
- Os rótulos só podem ser usados antes de do, while e for
- As chamadas só podem ocorrer dentro de blocos de repetição.

Exemplo:

```
revista: while (!terminado) {  
    for (int i = 10; i < 100; i += 10) {  
        passePagina();  
        if (textoChato) {  
            break revista;  
        }  
    }  
    maisInstrucoes();  
}  
restoDoPrograma();
```

break sem rótulo quebraria aqui!

- Sintaxe:

ident: **do** {...}

ou

ident: **while** () {...}

ou

ident: **for** () { ... }

switch (case)

- **Sintaxe** *qualquer expressão que resulte em valor inteiro (incl. char)*

```
switch (seletor_inteiro) {  
    case valor_inteiro_1 :  
        instruções ;  
        break ;  
    case valor_inteiro_2 :  
        instruções ;  
        break ;  
    ...  
    default :  
        instruções ;  
}
```

uma constante inteira (inclui char)

- **Exemplo**

```
char letra ;  
  
switch (letra) {  
    case 'A' :  
        System.out.println("A") ;  
        break ;  
    case 'B' :  
        System.out.println("B") ;  
        break ;  
    ...  
    default :  
        System.out.println("?") ;  
}
```


- 1. Escreva um programa *Quadrados* que leia um número da linha de comando e imprima o quadrado de todos os números entre 1 e o número passado.
 - Para converter de *String* para *int*, use:

```
int numero = Integer.parseInt("10");
```
- 2. Use o *JOptionPane* (veja documentação) e repita o exercício anterior recebendo os dados através da janela de entrada de dados (programa *WinQuadrados*)
 - Use `JOptionPane.showInputDialog(string)` de `javax.swing` para ler entrada de dados
 - Use `JOptionPane.showMessageDialog(null, msg)` para exibir a saída de dados
- 3. Se desejar, use o *build.xml* do Ant disponível que executa *Quadrados* e *WinQuadrados*

Curso J100: Java 2 Standard Edition

Revisão 17.0

© 1996-2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br