



# Paradigmas de Programação

## Expressões e Atribuições

Kellyton Brito  
kellyton.brito@gmail.com  
10/11/2017



# Introdução

- Expressões e atribuições são a base das linguagens imperativas e derivadas
- Expressões são a maneira fundamental de expressar programas
- Sua sintaxe pode variar



**UFRPE**

Universidade  
Federal Rural  
de Pernambuco



*Tela de Lula Cardoso Ayres*

# Expressões



# Tipos de Expressões

- Quais os tipos de expressões das linguagens?
- Expressões Aritméticas
- Expressões Relacionais
- Expressões Booleanas



# Tipos de Expressões

- Quais os tipos de expressões das linguagens?
- **Expressões Aritméticas**
- Expressões Relacionais
- Expressões Booleanas





# Expressões Aritméticas

- Avaliações aritméticas similares às matemáticas
- Ordem de avaliação de Operadores
  - Precedência
  - Associatividade
  - Parênteses
  - Condicionais



# Ordem de Avaliação de Operadores

- Precedência em Java

Operators	Precedence
postfix	<code>expr++ expr--</code>
unary	<code>++expr --expr +expr -expr ~ !</code>
multiplicative	<code>* / %</code>
additive	<code>+ -</code>
shift	<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>
relational	<code>&lt; &gt; &lt;= &gt;= instanceof</code>
equality	<code>== !=</code>
bitwise AND	<code>&amp;</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&amp;&amp;</code>
logical OR	<code>  </code>
ternary	<code>? :</code>
assignment	<code>= += -= *= /= %= &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>



# Ordem de Avaliação de Operadores

- Associação

- Quando duas ocorrências adjacentes tem a mesma precedência
- Normalmente feita da esquerda para direita

$$A - B + C = (A - B) + C$$

- Parênteses

- Usados para alterar a precedência





# Ex.: Precedência e Associativade em C/C++

	++ --	Incrementar(sufixo) Diminuição(sufixo)	<< i; for( int i = 10; i > 0; i-- ) cout << i;	sim sim	
5	* / %	Multiplicação Divisão Módulo	int i = 2 * 4; float f = 10.0 / 3.0; int rem = 4 % 3;	sim sim sim	esquerda para direita
6	+ -	Adição Subtração	int i = 2 + 3; int i = 5 - 1;	sim sim	esquerda para direita
7	<< >>	Bitwise shift à esquerda Bitwise shift à direita	int flags = 33 << 1; int flags = 33 >> 1;	sim sim	esquerda para direita
8	< <= > >=	Comparação menor-que Comparação maior-ou-igual-que Comparação maior-que Comparação maior-ou-igual-que	if( i < 42 ) ... if( i <= 42 ) ... if( i > 42 ) ... if( i >= 42 ) ...	sim sim sim sim	esquerda para direita
9	== eq != not_eq	Comparação igual a Outra maneira de chamar == Comparação diferente de Outra maneira de chamar !=	if( i == 42 ) ... if( i != 42 ) ...	sim - sim	left to right
10	& bitand	Bitwise AND Outra maneira de chamar &	flags = flags & 42;	sim	esquerda para direita
11	^ xor	Bitwise exclusive OR (XOR) Outra maneira de chamar ^	flags = flags ^ 42;	sim	esquerda para direita

- Fonte e referência completa:  
[http://www.cppreference.com/wiki/br-pt/operator\\_precedence](http://www.cppreference.com/wiki/br-pt/operator_precedence)





## Ordem de Avaliação de Operandos

- Avaliação normalmente trivial
  - Valores da memória associados às variáveis
- Casos especiais: Efeitos colaterais
  - O que fazer quando uma função altera o valor de uma variável que está na mesma operação?



## Ordem de Avaliação de Operandos

*O que fazer quando uma função altera o valor de uma variável que está na mesma operação?*

```
int a;  
int b;  
  
private void start() {  
    a = 10; b = 0;  
    b = a + mudaA();  
    System.out.println("A = " + a + ", B = " + b);  
  
    a = 10; b = 0;  
    b = mudaA() + a;  
    System.out.println("A = " + a + ", B = " + b);  
}  
  
private int mudaA() {  
    a = 100;  
    return 1000;  
}
```





# Ordem de Avaliação de Operandos

- Casos especiais: Efeitos colaterais
  - Função altera seus parâmetros ou uma variável global
- Duas soluções
  - Desativar os efeitos colaterais: ler todos os operandos antes de executar as funções
  - Definir a ordem de avaliação dos operandos





# Ordem de Avaliação de Operandos

Solução tradicional: avaliação sempre da esquerda para direita

```
int a;  
int b;
```

```
private void start() {  
    a = 10; b = 0;  
    b = a + mudaA();  
    System.out.println("A = " + a + ", B = " + b);  
  
    a = 10; b = 0;  
    b = mudaA() + a;  
    System.out.println("A = " + a + ", B = " + b);  
}
```

```
private int mudaA() {  
    a = 100;  
    return 1000;  
}
```

A = 100, B = 1010  
A = 100, B = 1100



# Conversão de Tipos em Expressões

- Já discutido em outras aulas
- Em OO, tradicionalmente:
  - Tipos *menores* são convertidos implicitamente para tipos *maiores*
  - Tipos “maiores” podem ser convertidos explicitamente (cast) para tipos menores

*float a = 0;*

*int b = 5;*

*a = b;*

*b = (int)a;*



# Conversão de Tipos em Expressões

- Coerção

- Quando um tipo é *promovido* para participar de uma operação

*int a;*

*float b, c, d;*

*...*

*d = b \* a;*

- *a* é *promovido* para float para poder participar da multiplicação



# Conversão de Tipos em Expressões

- Erros comuns: Divisão por zero e overflow
- Overflow?
  - Quando um número não pode ser representado por determinado tipo
  - Ex.:

```
byte a = 100;  
byte b = 100;  
byte c = a * b; // Não suporta
```



# Conversão de Tipos em Expressões

## • Exemplo em Java

```
private void start() {  
    byte a, b, c;  
    a = 10;  
    b = 100;  
    c = a * b;  
  
    System.out.println(c);  
}
```

Não compila:

Em operações aritméticas, Java faz coerção dos tipos numéricos menores do que **int**, para **int**. No exemplo,  $a * b$  retorna um inteiro

- a) Dá erro de compilação
- b) Dá erro de execução
- c) Imprime 1000
- d) Imprime um número positivo diferente de 1000
- e) Imprime um número negativo





# Conversão de Tipos em Expressões

- E agora?

```
private void start() {  
    byte a, b, c;  
    a = 10;  
    b = 100;  
    c = (byte)a * (byte)b;  
  
    System.out.println(c);  
}
```

Mesma situação anterior.

- a) Dá erro de compilação
- b) Dá erro de execução
- c) Imprime 1000
- d) Imprime um número positivo diferente de 1000
- e) Imprime um número negativo



# Conversão de Tipos em Expressões

- E agora?

```
private void start() {  
    byte a, b, c;  
    a = 10;  
    b = 100;  
    c = (byte) (a * b);  
  
    System.out.println(c);  
}
```

- a) Dá erro de compilação
- b) Dá erro de execução
- c) Imprime 1000
- d) Imprime um número positivo diferente de 1000
- e) Imprime um número negativo

**Imprime -24.**

Java faz a operação considerando *int*'s, que dá 1000. Depois é feito cast para byte -> truncado.

Em bytes, de 00000011 11101000 para 11101000, o que dá -24 (bit de sinal + 104)



# Conversão de Tipos em Expressões

- Solução?

```
private void start() {  
    byte a, b, c;  
    a = 10;  
    b = 100;  
    c = (byte) (a * b);  
  
    System.out.println(c);  
}
```

- a) Dá erro de compilação
- b) Dá erro de execução
- c) Imprime 1000
- d) Imprime um número positivo diferente de 1000
- e) Imprime um número negativo

Escolha os tipos adequados para as operações. Por exemplo, **c** como **int**.



# Tipos de Expressões

- Quais os tipos de expressões das linguagens?
- Expressões Aritméticas
- **Expressões Relacionais**
- Expressões Booleanas



# Expressões Relacionais

- Expressões relacionais:
  - Comparam os valores dos dois operandos
  - O resultado é booleano
  - Exemplo:

*C-based  
Languages*

==

!=

>

<

>=

<=





# Tipos de Expressões

- Quais os tipos de expressões definidos para a gramática do tutorial?
- Expressões Aritméticas
- Expressões Relacionais
- **Expressões Booleanas**



# Expressões Booleanas

- Expressões Booleanas
  - Produz valores booleanos a partir de valores booleanos e expressões relacionais
  - Normalmente inclui AND, OR, NOT e XOR
  - Precedência maior a AND do que OR



# Avaliação de Curto Circuito

- É quando não se precisa avaliar os dois lados para ter o resultado
- Ex.:

*if ((a > 0) && (b > 0))*

*Se a <= 0, não precisa avaliar b*

- No geral:

- && e ||: AND e OR com curto circuito
- & e |: AND e OR sem curto circuito

Quando usar sem curto circuito é importante?

- Quando a avaliação altera valores



**UFRPE**

Universidade  
Federal Rural  
de Pernambuco



*Tela de Lula Cardoso Ayres*

# Atribuições



# Atribuição

- Tarefa de alterar o valor de uma variável
- Atribuição simples:

<variável\_destino> <operador\_de\_atribuicao> <expressao>

Uso do = para atribuição, e == para comparação





## Exercício

- O que acontece se for executado o seguinte código:

$$A + B = C + D$$

Suponha  $A = 1$ ,  $B = 2$ ,  $C = 3$ ,  $D = 4$

\*Testem em casa



# Atribuição

- Atribuição condicional – Operador ternário

```
flag ? count1 : count2 = 0;
```

equivalente a

```
if (flag)
    count1 = 0;
else
    count2 = 0;
```



# Atribuição

- Atribuição combinada

- O que significam os operadores:

$+=$      $-=$      $*=$      $/=$      $\%=$

**$a += b$  é equivalente a  $a = a + b$**

**$a *= b$  é equivalente a  $a = a * b$**

***E assim sucessivamente***



# Atribuição

- Operadores unários
  - Qual a diferença entre:

*a = count ++*

*e*

*a = ++count*



# Atribuição

- Operadores unários

Operador	Equivalente a
<code>sum = ++count</code>	<code>count = count + 1</code> <code>sum = count</code>
<code>sum = count++</code>	<code>sum = count</code> <code>count = count + 1</code>





# Atribuição

- Quais os valores impressos ao ser executado esse código?

```
private void start2 () {  
    int a = 0;  
    int i = 0;  
  
    while (++i < 5) {  
        a += i++;  
        System.out.println("Intermediário: A = " + a + ", I = " + i);  
    }  
    System.out.println("Final: A = " + a + ", I = " + i);  
}
```

Intermediário: A = 1, I = 2

Intermediário: A = 4, I = 4

Final: A = 4, I = 5



# Atribuição

- Atribuição como expressão
  - Quando uma atribuição faz parte de uma expressão

```
while ((ch = getchar()) != EOF) { ... }
```

- C permite testar a atribuição, e não apenas o resultado
  - if (a = b) → normalmente verdadeiro*
- Normalmente não se permite o teste apenas da atribuição
  - Evita erros por parte do programador



## Atribuição em modo misto

- Quando o tipo da expressão não é o mesmo da variável que foi atribuída
- Se a linguagem faz *coerção* (promoção) dos operandos da expressão

- Ex.:

```
int a, b;  
float c;  
...  
c = a / b;
```

*A e B sofrem coerção para tipos float, mas devolve para int antes da atribuição*



# Dúvidas quanto a expressões e atribuições?

Sebesta – Conceitos de Linguagens de Programação. Até o capítulo 7