

INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO Java

Objetivos

- Apresentar a descrição da linguagem Java;
- Apresentar as estruturas básicas de controle em Java;
- Apresentar a forma de codificação em linguagem Java;
- Apresentar padrões de mapeamento para a linguagem Java.

Histórico

- 1995 – primeira versão de Java anunciada pela Sun;
- 1996 – publicação dos livros "The Java Tutorial" e "Java Language Specification";
- 1998 – lançamento da plataforma Java 2;
- 1999 – lançamento das plataformas J2SE, J2EE e J2ME.

Descrição da linguagem

- Alfabeto

Um programa em Java poderá conter os seguintes caracteres:

- as vinte e seis (26) letras do alfabeto inglês:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
- os dez (10) algarismos:
0 1 2 3 4 5 6 7 8 9
- os símbolos:

| | | | |
|---|------------------------------|-----|------------|
| < | menor | () | parênteses |
| > | maior | [] | colchete |
| . | ponto | { } | chaves |
| , | vírgula | + | soma |
| : | dois pontos | - | subtração |
| ; | ponto-e-vírgula | * | asterisco |
| = | igualdade | / | barra |
| ! | exclamação | | |
| ? | interrogação | “ ” | aspas |
| & | ampersete ("e" comercial) | ‘ ’ | apóstrofo |
| ^ | circunflexo | % | porcento |
| | barra em pé | ~ | til |

- Pontuação
 - Ponto-e-vírgula é usado para separar comandos, a menos que outro separador seja necessário;
 - Em alguns casos de operadores, convém o uso de espaços em branco antes, e depois.
- Observação:
Em Java utilizam-se, **obrigatoriamente**, as letras minúsculas para os comandos próprios da linguagem. Por convenção, armazenadores de dados e métodos também começam por minúsculas e nomes de classes por maiúsculas.

Tipos de dados

- Tipos básicos

| Algoritmo | Java |
|-----------|----------------|
| inteiro | int |
| real | double |
| caractere | char |
| lógico | boolean |

- Outros tipos:

| | |
|--------------|----------------------------------|
| byte | inteiros "mais curtos" (08 bits) |
| short | inteiros "curtos" (16 bits) |
| long | inteiros "compridos" (64 bits) |
| float | reais com precisão simples |

- Especificação de modo de armazenamento:

| | |
|---------------|-----------------------------------|
| final | constante |
| static | alocação estática |
| public | acessível durante toda a execução |

Exemplos:

```
static int x;
final double pi = 3.1415;
public char espaco = ' ';
public static double y = 0.0;
public final static int zero = 0
```

| Resumo | Java | Tipos de dados primitivos | Tabela de referência |
|----------------|------|----------------------------------|-------------------------------------|
| | bits | Valor mínimo | Valor máximo |
| boolean | 1 | false / true | |
| byte | 8 | -128 | +127 |
| char | 16 | Unicode (0) | Unicode ($2^{16}-1 = 65535$) |
| short | 16 | $-2^{15} = -32768$ | $2^{15} - 1 = +32767$ |
| int | 32 | $-2^{31} = -2147483648$ | $2^{31} - 1 = +2147483647$ |
| long | 64 | $-2^{63} = -9223372036854775808$ | $2^{63} - 1 = +9223372036854775807$ |
| float | 32 | -10^{38} (IEEE-754) | 10^{38} (IEEE-754) |
| double | 64 | -10^{308} (IEEE-754) | 10^{308} (IEEE-754) |

- Constantes

- Constante inteira

Exemplos:

| | |
|------------------------|-----------------|
| 10, 532, -10567 | |
| 067, 05421, 0724 | (octal) |
| 0L, 1300000000L, 3241L | (inteiro longo) |
| 0x41, 0xFFFF, 0xA0 | (hexadecimal) |

- Constante real

Exemplos:

10.465 -5.61 +265. 0.0 .731 .37e+2 -3.e-1

Observações:

A vírgula decimal é representada por ponto decimal.

- Constante literal

Exemplos:

| | |
|-----------|--------------------------------|
| Caractere | : '1', ' ', '*', 'A', 'a', '?' |
| Cadeia | : "BRANCO", "CEU AZUL" |

- Caracteres pré-definidos:

| | |
|--------|-------------------------------------------------------|
| '\0' | nulo (fim de cadeia de caracteres) |
| '\n' | passa para a próxima linha |
| '\t' | passa para a próxima coluna de tabulação (9,17, ...) |
| '\b' | retorna o cursor uma coluna |
| '\r' | posiciona o cursor no início da linha |
| '\f' | limpa a tela ou passa para a próxima página |
| '\\' | barra invertida |
| '\"' | apóstrofo |
| '\nnn' | representação de um byte , em octal |
| '\xnn' | representação de um byte , em hexadecimal |

- Definição de constantes

Forma geral:

final <tipo> <NOME> = <valor>;

Exemplos:

static final double pi = 3.1415926;

- Armazenadores de dados

- Nome de armazenador

- a) O nome de um armazenador tem tamanho determinado;
 - b) O primeiro caractere é uma letra ou travessão (_);
 - c) Outros caracteres podem ser letra, algarismo ou travessão (_).

Exemplos:

Nomes válidos : x, a, de, v9a, lista_Notas

Nomes inválidos: x+, t.6, 43x, so 5

- Definição de armazenadores

- Armazenadores simples

Forma geral:

```
<tipo 1> <lista de nomes 1>;  
<tipo 2> <lista de nomes 2>;  
...  
<tipo N> <lista de nomes N>;
```

Exemplos:

```
char    fruta;  
int     i, j, k;  
double  p, delta;
```

A definição de armazenadores pode ser usada para atribuir valores iniciais.

Exemplos:

```
int  x = 10,  
      y = 20;
```

- Armazenadores agrupados
- Homogêneos

Forma geral:

```
<tipo 1> [ ] <lista de nomes 1> = new <tipo 1> [ tamanho ];
<tipo 2> [ ] [ ] <lista de nomes 2> = new <tipo 1> [ tamanho 1 ] [ tamanho 2 ];
...
<tipo N> <lista de nomes N> [índice] ... ;
```

Exemplos:

```
char [ ] frutas = new char [10],
char [ ] [ ] nomes = new char [3][10];
int [ ] v = new int [5];
```

Observação:

O primeiro elemento tem índice igual a zero.

- Heterogêneas

Forma geral:

```
class <nome> {<definições>}
```

```
<nome> <objeto>;
<nome> <objeto 1>, <objeto 2>;
```

Exemplos:

```
class Complexo
{
    double real, imag;
}
```

```
Complexo x, y;
```

Observação:

O acesso aos componentes de uma classe pode ser feito por

```
nome.membro
```

- Tipos de operadores
 - Aritméticos

| Algoritmo | Java |
|----------------|-------|
| * / mod | * / % |
| + - | + - |

Observações:

O operador **div** (divisão inteira) é a própria barra (/), quando os operandos forem inteiros.

Existem formas compactas para incremento e decremento:

| | |
|----------------------|----------------|
| <variável inteira>++ | pós-incremento |
| ++<variável inteira> | pré-incremento |
| <variável inteira>-- | pós-decremento |
| --<variável inteira> | pré-decremento |

- Relacionais

| Algoritmo | Java |
|-----------|-----------|
| < ≤ > ≥ | < <= > >= |
| = ≠ | == != |

Observação:

O resultado de uma comparação de dois valores pode ser **false** (falso) ou **true** (verdadeiro).

- Lógicos (bit a bit)

| Algoritmo | Java |
|-------------------------|------|
| complemento de um | ~ |
| e | & |
| ou-exclusivo | ^ |
| ou | |
| deslocamento à direita | >> |
| deslocamento à esquerda | << |

Observação:

O resultado de uma operação lógica é um valor cujos bits são operados um a um de acordo com a álgebra de proposições.

- Conectivos lógicos

| Algoritmo | Java |
|-----------|------|
| não | ! |
| e | && |
| ou | |

- Prioridade de operadores

| Operador | Associação |
|------------------------|------------|
| () [] | à esquerda |
| ! ~ ++ -- + - (tipo) & | à direita |
| * / % | à esquerda |
| + - | à esquerda |
| >> << | à esquerda |
| < <= >= > | à esquerda |
| == != | à esquerda |
| & | à esquerda |
| ^ | à esquerda |
| | à esquerda |
| && | à esquerda |
| | à esquerda |
| ?: | à direita |
| = += -= *= /= %= | à direita |
| >>= <<= &= = ^= | à direita |
| , | à esquerda |

- Funções intrínsecas

As regras usadas na formação dos nomes dessas funções intrínsecas são as mesmas utilizadas para os nomes de variáveis.

Exemplo:

`a = sin (b)`

a - nome da variável que receberá o resultado da função;
 sin - função (seno) predefinida do Java ;
 b - nome da variável que vai ser o argumento da função.

| Nome (argumento) | Tipo de argumento | Descrição |
|------------------|-----------------------|------------------------|
| Math.sin (X) | double | seno (em radianos) |
| Math.cos (X) | double | cosseno (em radianos) |
| Math.tan (X) | double | tangente (em radianos) |
| Math.asin(X) | double | arco seno |
| Math.acos(X) | double | arco cosseno |
| Math.atan(X) | double | arco tangente |
| Math.sqrt(X) | double | raiz quadrada |
| Math.exp (X) | double | exponencial de "e" |
| Math.abs (X) | numérico | valor absoluto |
| Math.log (X) | double | logaritmo neperiano |
| Math.pow(X,Y) | double, double | eleva X a Y |

A linguagem Java dispõe de uma significativa biblioteca básica (**Math**), com diversas outras funções além das aritméticas citadas acima.

- Expressões

- Aritmética

Exemplos:

| Algoritmo | Java |
|-----------------------|-------------------------|
| 10 + 15 | 10 + 15 |
| 543.15/3 | 543.15/3 |
| $(x + y + z) * a / z$ | $((x + y + z) * a) / z$ |

- Lógica

Exemplos:

| Algoritmo | Java |
|----------------------------|------------------------------|
| A = 0 | A == 0 |
| a ≠ 1 | a != 1 |
| $(A \geq 0) \& (a \leq 1)$ | $(A \geq 0) \&\& (a \leq 1)$ |

Observação:

Para efeito de clareza, ou para mudar a precedência de operadores, pode-se separar as proposições por parênteses.

- Estrutura de programa

// definições para pré-processamento

class <nome>

{

// definições globais

// definições de armazenadores e métodos (*OPCIONAL*)

<tipo> <nome> (<lista de parâmetros>)

{

// definições locais

// comandos

}

// parte principal

public static void main (**String** [] args)

{

// definições locais

// comandos

}

} // fim da classe

- Comentários

Comentários são precedidos pelos sinais `//`, `/* */` ou `/** */` envolvendo o texto.

Exemplo:

```
/**
 * Comentario para documentacao automatica do metodo
 */
public static void main ( String [ ] args )
{
    // Este programa nao faz nada - comentario

    /*
    que tambem pode ser colocado assim, sem ser automatico
    */
}
```

- Atribuição

- Atribuição simples

Forma geral:

`<variável> = <expressão>;`

Exemplo:

```
x    = 0 ;
a    = 1.57;
letra = 'A' ;
```

- Atribuição múltipla

Forma geral:

`<variável 1> = <variável 2> = <expressão>;`

Exemplo:

```
x = y = 0;
```

Observação:

A execução inicia-se pela direita.

- Atribuição composta

Forma geral:

`<variável> <operador> = <expressão>;`

Exemplo:

```
i += 1   ou   i = i + 1
```

Observação:

Operadores permitidos: `+` `-` `*` `/` `%` `>>` `<<` `|` `&` `^`

- Atribuição condicional

Forma geral:

```
<variável> = <teste> ? <expressão 1>: <expressão 2>;
```

Exemplo:

```
x = (a < b) ? a: b;    // se verdadeiro então ( a ), senão ( b )
```

- Definição de entrada e saída
- Entrada/saída (padrão Java):

Forma geral:

```
System.in.read    ( );
System.out.print  ( <valor> );
System.out.println ( <valor> );
```

Observação:

É necessário usar a definição de pacote abaixo (ou similar):

```
import java.io.*;
```

As operações de entradas de dados devem lidar com a possibilidade de ocorrência de erros. Recomenda-se, por isso, que sejam realizadas dentro de regiões críticas, com o devido tratamento de exceções.

As operações para saída de valores preferencialmente lidam com cadeias de caracteres (**String**).

- Caracteres com funções especiais:

| caractere | função |
|-----------|------------------------------|
| \0 | fim da cadeia de caracteres |
| \n | fim de linha (LF) |
| \t | tabulação |
| \b | retrocesso (BS) |
| \r | retorno de carro (CR) |
| \f | avanço de carro (FF) |
| \\ | barra invertida |
| \' | apóstrofo |
| \xnn | representação em hexadecimal |

Exemplo completo de programa:

```

/**
 *   Exemplo class
 */
// Inclusao de classes úteis para entrada e saida
import java.io.*;
import java.util.*;

class Exemplo
{
    private static BufferedReader console
        = new BufferedReader ( new InputStreamReader ( System.in ) );

    public static void main ( String [ ] args )
    {
        int    x, y;
        String linha;

        System.out.println ( "Exemplo: " );
        System.out.println ( );
        System.out.println ( "Digitar um numero inteiro: " );
        try                                // região crítica (pode ocorrer erro fatal)
        {
            linha= console.readLine ( );    // tentar ler uma linha
        }
        catch ( IOException IOEx )        // caso ocorra erro, tratar exceção
        {                                  // indicar o erro
            System.out.println ( "\nERRO: leitura." );
            IOEx.printStackTrace ( );
        }                                // fim da região crítica
        // se a leitura der certo, converter para inteiro também pode gerar exceção
        y = Integer.parseInt ( linha );    // recomenda-se usar região crítica
        x  = y * 2 + 10;
        System.out.println ( "\nO resultado e' igual a " + x + "." );
    } // fim main ( )
} // fim da classe

```

Se fornecido o valor 5 para a variável **y** , o resultado será:

O resultado e' igual a 20.

- Estruturas de controle
- Sequência simples

Forma geral:

| Algoritmo | Java |
|-----------|-------------|
| <comando> | <comando> ; |
| <comando> | <comando> ; |

Observação:

Em Java todos os comandos são separados por ponto-e-vírgula.

- Estrutura alternativa
- Alternativa simples

Forma geral:

| Algoritmo | Java |
|---------------|------------------------|
| se <condição> | if (<condição>) |
| então | { |
| <comandos> | <comandos> ; |
| fim se | } |

- Alternativa dupla

Forma geral:

| Algoritmo | Java |
|---------------------|------------------------|
| se <condição> então | if (<condição>) |
| <comandos 1> | { <comandos 1> ; } |
| senão | else |
| <comandos 2> | { <comandos 2> ; } |
| fim se | |

- Alternativa múltipla

Forma geral:

| Algoritmo | Java |
|------------------|-----------------------|
| escolher <valor> | switch <valor> |
| | { |
| <opção 1>: | case 1: |
| <comandos 1> | <comandos 1> ; |
| | break ; |
| <opção 2>: | case 2: |
| <comandos 2> | <comandos 2> ; |
| | break ; |
| ... | ... |
| <opção n-1>: | case (n-1): |
| <comandos N-1> | <comandos N-1> ; |
| | break ; |
| senão | default : |
| <comandos N> | <comandos N> |
| fim escolher | } |

Observações:

A variável de decisão deve ser escalar (enumerável, com certa ordem).

Se o comando **break** for omitido, os comandos da opção seguinte também serão executados.

A indicação **default** é opcional.

- Estrutura repetitiva

- Repetição com teste no início

Forma geral:

| Algoritmo | Java |
|-----------------------------|---------------------------|
| repetir enquanto <condição> | while (<condição>) |
| <comandos> | { |
| fim repetir | <comandos> ; |
| | } |

Observação:

A condição para execução é sempre verdadeira.

- Repetição com teste no início e variação

Forma geral:

| Algoritmo | Java |
|------------------------------|-------------------|
| repetir para | for (|
| <variável> = <valor inicial> | <valor inicial> ; |
| : <fim> | <teste de fim> ; |
| : <variação> | <variação>) |
| <comandos> | { |
| fim repetir | <comandos> ; |
| | } |

Observações:

A condição para execução é sempre verdadeira.

Em Java, qualquer um dos elementos, ou mesmo todos, podem ser omitidos. Entretanto, se tal for preciso, recomenda-se o uso de outra estrutura mais apropriada.

- Repetição com teste no fim

Forma geral:

| Algoritmo | Java |
|------------------------|-------------------------------|
| repetir até <condição> | do |
| <comandos> | { |
| fim repetir | <comandos> ; |
| | } while (<condição>) ; |

Observação:

A condição para execução é sempre verdadeira.

- Interrupções

Em Java, as repetições podem ser interrompidas, em sua sequência normal de execução através dos comandos:

break; e **continue;**

O comando **break** serve para interromper completamente uma repetição, passando o controle ao próximo comando, após a estrutura repetitiva.

O comando **continue** interrompe uma iteração, voltando ao início.