Apostila de JAVA

(versão reduzida)

Grupo PET – Informática

Sumário

1. INTRODUÇÃO

1.1. O que é JAVA?

Java é uma linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems. Modelada depois de C++, a linguagem Java foi projetada para ser pequena, simples e portável a todas as plataformas e sistemas operacionais, tanto o código fonte como os binários. Esta portabilidade é obtida pelo fato da linguagem ser interpretada, ou seja, o compilador gera um código independente de máquina chamado *bytecode*. No momento da execução este *bytecode* é interpretado por uma máquina virtual instalado na máquina. Para portar Java para uma arquitetura de hardware específica, basta instalar a máquina virtual (interpretador). Além de ser integrada à Internet, Java também é uma excelente linguagem para desenvolvimento de aplicações em geral. Dá suporte ao desenvolvimento de software em larga escala.

1.2. Criando uma APLICAÇÃO

Para começar, criaremos uma simples aplicação em Java: a clássica "Hello World!", o exemplo que todos os livros de linguagens usam.

1.2.1. O código fonte

Como todas as linguagens de programação, o código fonte será criado em um editor de texto ASCII puro. No Unix alguns exemplos são emacs, pico, vi e outros. No Windows, notepad ou dosedit também servem.
A seguir, o código da aplicação "Hello World!" (arquivo: HelloWorld.java):

```
class HelloWorld
{
      public static void main(String args[])
      {
            System.out.println("Hello World!");
      }
}
```

1.2.2. Compilando a aplicação

Para compilar a aplicação, basta digitar o comando:

javac HelloWorld.java

Este comando vai gerar o arquivo HelloWorld.class, que é o *bytecode* da aplicação. Para executar o *bytecode* basta digitar o comando:

java HelloWorld

2. O BÁSICO

2.1. Variáveis e tipos de dados

Variáveis são alocações de memória nas quais podemos guardar dados. Elas têm um nome, tipo e valor. Toda vez que necessite usar de uma variável você precisa declará-la e só então poderá atribuir valores a mesma.

2.1.1. Declarando variáveis

As declarações de variáveis consistem de um tipo e um nome de variável: como segue o exemplo:

int idade;
String nome;
boolean existe;

Os nomes de variáveis podem começar com uma letra, um sublinhado (_), ou um cifrão ($). Elas não podem começar com um número. Depois do primeiro caracter pode-se colocar qualquer letra ou número.

2.1.2. Tipos de variáveis

Toda variável deve possuir um tipo. Os tipos que uma variável pode assumir uma das três "coisas" a seguir:

- Uma das oito primitivas básicas de tipos de dados
- O nome de uma classe ou interface
- Um Array

Veremos mais sobre o uso de arrays e classes mais a frente.
Os oito tipos de dados básicos são: inteiros, números de ponto-flutuante, caracteres e booleanos (verdadeiro ou falso).
Tipos Inteiros:

Tipo	Tamanho	Alcance
byte	8 bits	-128 até 127
short	16 bits	-32.768 até 32.767
int	32 bits	-2.147.483.648 até 2.147.483.647
long	64 bits	-9223372036854775808 até 9223372036854775807

Existem dois tipos de números de ponto-flutuante: float (32 bits, precisão simples) e double (64 bits, precisão dupla).

2.1.3. Atribuições a variáveis

Após declarada uma variável a atribuição é feita simplesmente usando o operador '=':

idade = 18;
existe = true;

2.2. Comentários

Java possui três tipos de comentário, o /* e */ como no C e C++. Tudo que estiver entre os dois delimitadores são ignorados:

```
/* Este comentário ficará visível somente no código o compilador ignorará
completamente este trecho entre os delimitadores
*/
```

Duas barras (//) também podem ser usadas para se comentar uma linha:

```
int idade; // este comando declara a variável idade
```

E finalmente os comentários podem começar também com /** e terminar com */. Este comentário é especial e é usado pelo ***javadoc*** e para gerar uma documentação API do código. Para aprender mais sobre o ***javadoc*** acesse a home page (http://www.javasoft.com).

2.3. Caracteres especiais

Caracter	Significado
\n	Nova Linha
\t	Tab
\b	Backspace
\r	Retorno do Carro
\f	"Formfeed" (avança página na impressora)
\\	Barra invertida
\'	Apóstrofe
\"	Aspas
\ddd	Octal
\xdd	Hexadecimal

2.4. Expressões e operadores

2.4.1. Operadores Aritméticos

Operador	Significado	Exemplo
+	soma	3 + 4
-	subtração	5 - 7
*	multiplicação	5 * 5
/	divisão	14 / 7
%	módulo	20 % 7

Exemplo Aritmético:

```
class ArithmeticTest
{
      public static void main(Strings args[])
      {
            short x = 6;
            int y = 4;
            float a = 12.5f;
            float b = 7f;

            System.out.println ( "x é " + x + ", y é " + y );
            System.out.println ( "x + y = " + (x + y) );
            System.out.println ( "x - y = " + (x - y) );
            System.out.println ( "x / y = " + (x / y) );
            System.out.println ( "x % y = " + ( x % y ) );

            System.out.println ( "a é " + a + ", b é " + b );
            System.out.println ( " a / b = " + ( a / b ) );
      }
}
```

A saída do programa acima é :

```
x é 6, y é 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
```

```
a é 12.5, b é 7
a / b = 1.78571
```

2.4.2. Mais sobre atribuições

Variáveis podem atribuídas em forma de expressões como:

int x, y, z;

x = y = z = 0;

No exemplo as três variáveis recebem o valor 0;

Operadores de Atribuição:

Expressão	Significado
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y

2.4.3. Incrementos e decrementos

Como no C e no C++ o Java também possui incrementadores e decrementadores :

y = x++;

y = --x;

As duas expressões dão resultados diferentes, pois existe uma diferença entre prefixo e sufixo. Quando se usa os operadores (x++ ou x--), y recebe o valor de x antes de x ser incrementado, e usando o prefixo (++x ou –x) acontece o contrario, y recebe o valor incrementado de x.

2.5. Comparações

Java possui várias expressões para testar igualdade e magnitude. Todas as expressões retornam um valor booleano (true ou false).

2.5.1. Operadores de comparação

Operador	Significado	Exemplo
==	Igual	x == 3
!=	Diferente (Não igual)	x != 3
<	Menor que	x < 3
>	Maior que	x > 3
<=	Menor ou igual	x <= 3
>=	Maior ou igual	x >= 3

2.5.2. Operadores lógicos

Operador	Significado
&&	Operação lógica E (AND)
||	Operação lógica OU (OR)
!	Negação lógica
&	Comparação bit-a-bit E (AND)
|	Comparação bit-a-bit OU (OR)
^	Comparação bit-a-bit OU-Exclusivo (XOR)
<<	Deslocamento a esquerda
>>	Deslocamento a direita
>>>	Deslocamento a direita com preenchimento de zeros
~	Complemento bit-a-bit
x <<= y	Atribuição com deslocamento a esquerda (x = x << y)
x >>= y	Atribuição com deslocamento a direita (x = x >> y)

x >>>= y	Atribuição com deslocamento a direita e com preenchimento de zeros (x = x >>> y)
x &= y	atribuição AND (x = x & y)
x |= y	atribuição OR (x = x | y)
x ^= y	atribuição XOR (x = x ^ y)

3. ARRAYS, LOOPS E CONDICIONAIS

3.1. Arrays

Arrays em Java são diferentes do que em outras linguagens. Arrays em Java são objetos que podem ser passados e acoplados a outros objetos.

Arrays podem conter qualquer tipo de elemento com valor (tipos primitivos ou objetos), mas você não pode armazenar diferente tipos em um simples array.

Ou seja, você pode ter um array de inteiros, ou um array de strings, ou um array de array, mas você não pode ter um array que contenha ambos os objetos strings e inteiros.

A restrição acima descrita significa que os arrays implementados em Java são genéricos homogêneos, ou seja, um único array pode armazenar qualquer tipo de objeto com a restrição que todos sejam do mesma classe.

3.1.1. Declarando um Array:

```
String[] difficult;
Point[] hits;
int[] temp;
```

3.1.2. Criando Objetos Arrays:

Um dos caminhos é usar o operador *new* para criar uma nova instância de um array, por exemplo:

```
int[] temps = new int[99];
```

Quando você cria um objeto array usando o operador *new*, todos os índices são inicializados para você (0 para arrays numéricos, falso para boolean, '\0' para caracteres, e NULL para objetos). Você também pode criar e inicializar um array ao mesmo tempo.

```
String[] chiles = {"jalapeno", "anaheim", "serrano", "jumbou", "thai"};
```

Cada um dos elementos internos deve ser do mesmo tipo e deve ser também do mesmo tipo que a variável que armazena o array. O exemplo acima cria um array de Strings chamado *chiles* que contém 5 elementos.

3.1.3. Acessando os Elementos do Array

Uma vez que você têm um array com valores iniciais, você pode testar e mudar os valores em cada índice de cada array.

Os arrays em Java sempre iniciam-se na posição 0 como no C++. Por exemplo:

```
String[] arr = new String[10];
arr[10] = "out";
```

Isto provoca um erro de compilação pois o índice 10 não existe, pois isto está fora das bordas do array.

```
arr[9] = "inside";
```

Esta operação de atribuição é válida e insere na posição 9 do array, a string "*inside*".

3.1.4. Arrays Multidimensionais

Java não suporta arrays multidimensionais. No entanto, você pode declarar e criar um array de arrays e acessá-los como você faria no estilo-C.

```
int coords[][] = new int[12][12];
coords[0][0] = 1;
coords[0][1] = 2;
```

3.2. Condicionais

O condicional contém a palavra chave *if*, seguido por um teste booleano. Um opcional *else* como palavra chave pode ser executado na caso do teste ser falso, Exemplo:

```
if (x < y)
{
      System.out.println(" x e menor do que y");
}
else
{
      System.out.println(" y e maior);
}
```

Nota técnica: A diferença entre o if em Java e C ou C++ é que o teste deve retornar um valor booleano (true ou false).

3.2.1. Bloco

Um bloco é definido por ({}) e contém um grupo de outros blocos. Quando um novo bloco é criado um novo escopo local é aberto e permite a definição de variáveis locais. As variáveis definidas dentro de um bloco só podem ser vistas internamente a este, e são terminadas ou extintas no final da execução deste(}).

```
void testblock()
{
      int x = 10, w = 1;

      if (x > w)
      { // inicio do bloco
            int y=50;
            System.out.println("dentro do bloco");
            System.out.println("x:" + x);
            System.out.println("y:" + y);
      } // final do bloco
      System.out.println("w:" + w);
      System.out.println("y:" + y);  // erro variável não conhecida
}
```

3.3. O operador Condicional

Uma alternativa para o uso do *if* e *else* é um operador ternário condicional. Este operador ternário (*?:)* , é chamado assim porque tem três termos como parâmetro.

Exemplo:

```
test ? trueresult : falseresult
int m = x < y ? x : y ;  // a variável m recebe o valor do menor entre x e y
```

3.4. O switch

Um comum mecanismo para substituição de *ifs* que pode ser usado para um grupo de testes e ações junto a um simples agrupamento, chama-se *switch.*

```
switch (teste)
{
case valorum;
      resultum;
      break;
case valordois;
      resultdois;
      break;
case valortres:
      resulttres;
      break;
default: defaultresult;
}
```

O valor é comparado com cada um dos casos relacionados. Se a combinação não for encontrada, o bloco *default* executado. O *default* é opcional, então caso este não esteja associado ao comando, o bloco do *swicth* sem executar nada.

3.5. Loop For

O loop em Java tem esta sintaxe:

```
for (inicialização; teste; incremento)
{
      bloco de comandos;
}
```

Você também pode incluir um *comando* simples, sendo assim não há necessidade da utilização de chaves. Exemplo:

```
String[] strArray = new String[10];
for (i=0; i< strArray.length; i++)
{
      strArray[i] = "";
}
```

Inicializa um array de 10 elementos com *"";*

3.6. Loop While

O *while* é usado para repetir um comando, ou um conjunto de comando enquanto a condição é verdadeira.

```
while (condição)
{
      bloco de comandos;
}
```

A *condição* é uma expressão booleana. Exemplo:

```
int count = 0;
while ((count < array1.length) && (array1[count] != 0))
{
      array2[count] = (float) array1[count++];
}
```

3.7. Loop Do

A principal diferença entre o *while* e o *do* é que o teste condicional no caso do *while* é feita antes de se executar o código interno ao loop. Desta forma, o que pode acontecer no *while* é que o loop pode não ser executado se a condição for *false.* Já no loop *do* o corpo do loop é executado pelo menos uma vez, pois o teste de permanência é executado no fim do loop.

```
do
{
      bodyOfLoop;
} while (condition);
```