Hortolândia, 21 de agosto de 2013.

Aulas 15 e 16 - Declaração de Variáveis e Estruturas de Controle

Introdução

Esta é a segunda aula de **Introdução à Linguagem Java**.

Em aulas anteriores vimos o conceito de **tipos de dados primitivos**. Foram também apresentados os tipos primitivos em Java, bem como a classe *String* que, em função de sua simplicidade, é comparável a um tipo primitivo da linguagem.

Nas **Aulas 13 e 14** vimos como são implementados **vetores** e **matrizes** em Java.

Na presente aula (**Aulas 15 e 16**) serão vistos os assuntos de **declaração de variáveis** e **estruturas de controle**.

Em Java toda variável que for declarada precisa ser associada, logo no momento de sua declaração, a um determinado tipo de dados. As **declarações de variáveis** são o primeiro tópico da presente aula - **Seção 1**. Em seguida, na **Seção 2**, demonstra-se como realizar a atribuição de valores a variáveis.

Na **Seção 3** estudaremos os operadores, que são elementos que possibilitam ao programador a realização de comparações e operações entre expressões constituídas tanto por variáveis exclusivamente como por variáveis e constantes.

Já na **Seção 4** apresentaremos o último tópico da presente aula, correspondente às **estruturas de fluxo de controle**, as quais permitem ao programador traduzir a lógica de seus algoritmos em um programa da linguagem Java.

1. Declarações de Variáveis

Conforme visto na aula passada, em Java e em qualquer outra linguagem de **tipificação forte**, as **variáveis** antes de serem utilizadas precisam estar associadas a um determinado **tipo de dados**.

Esta associação é feita logo na primeira instrução que contenha o nome da variável, instrução esta chamada de declaração da variável.

Uma instrução de declaração de variável é uma sentença da linguagem que contém o tipo da variável e o seu nome, como nos exemplos a seguir.

```
double salario;

int diasDeFerias;

long populacaoDaTerra;

char caractere;

boolean pronto;

String mensagemAoUsuario;
```

Os termos em negrito correspondem aos tipos (no caso, os cinco primeiros são tipos primitivos e o sexto é o da classe String).

Uma convenção da linguagem com relação aos nomes de variáveis é que eles tenham sempre a primeira letra em minúsculo, e as iniciais de cada nova palavra justaposta à primeira estejam em maiúsculo, como no exemplo “populacaoDaTerra”.

Ao final de toda declaração precisa-se utilizar o caractere de ponto-e-vírgula.

Uma vez que a variável tenha sido declarada, pode-se utilizá-la para o armazenamento de valores do tipo que ela corresponde. A atribuição de valores a variáveis é o assunto da próxima seção.

2. Atribuição de valores a variáveis

A operação de atribuição de valores a variáveis é realizada por meio do uso do operador de atribuição "**=**", o qual separa o termo à sua esquerda (variável que receberá o valor atribuído) do termo à direita – valor a ser atribuído à variável.

Supondo que queiramos criar uma variável que contenha o número de dias que um determinado funcionário poderá ficar de férias. Tal variável pode ter o nome de "diasDeFerias", e o seu valor pode ser representado por um **int**. Então a sua instrução de declaração pode ter a forma:

```
int diasDeFerias;
```

Após a instrução de declaração, a variável já pode ser utilizada na seqüência do programa. A atribuição do valor de 12 dias a esta variável pode ser realizada com a seguinte instrução:

```
diasDeFerias = 12;
```

Costuma-se chamar a primeira instrução de atribuição de valor a uma variável de inicialização da variável. A inicialização corresponde, portanto, à atribuição de um valor inicial para esta variável. No decorrer do programa, outros valores podem ser atribuídos à mesma.

Existe a possibilidade de se realizar a declaração e a inicialização de uma variável em uma única linha, como no exemplo:

```
int diasDeFerias = 12;
```

A próxima seção trata do uso de operadores, dos quais já vimos o operador de atribuição "=".

3. Operadores

Existem diversos tipos de operadores em Java. O **operador de atribuição**, visto na seção anterior, possibilita a atribuição de valores a variáveis.

Estudaremos na presente seção os **operadores aritméticos**, os **relacionais** e os **lógicos** (ou **booleanos**).

3.1. Operadores Aritméticos

Os operadores aritméticos "+","-", "*" e "/" são utilizados para a adição, subtração, multiplicação e divisão, respectivamente.

Pode-se utilizar operadores aritméticos para a confecção de expressões, cujo valor pode ser então, após calculado pelo interpretador Java, armazenado em uma variável.

Os exemplos a seguir ilustram a utilização de operadores aritméticos em expressões, e também a atribuição dos valores destas expressões a variáveis.

Exemplo 1: Expressão de soma de dois operandos, e atribuição do valor resultante a uma variável "soma", declarada na instrução anterior.

```
int soma;

soma = 1 + 3;
```

Exemplo 2: Atribuição de valores a duas variáveis, e posterior operação de multiplicação de seus valores, armazenando o resultado em uma variável "resultado".

```
int operando1 = 5;

int operando2 = 7;

int resultado = operando1 * operando2;
```

Além dos operadores de soma, subtração, multiplicação e divisão descritos, também são operadores aritméticos os operadores de incremento e decremento.

Tais operadores servem para somar ou subtrair uma unidade a uma variável do tipo inteiro, e simplificam a escrita de uma operação de incremento ou decremento.

O **operador de incremento** é representado pela seqüência de caracteres "**++**", a qual pode estar à esquerda ou à direita de uma variável do tipo int.

Se estiver à esquerda, o operador indica a operação de pré-incremento, ou seja, o operando é primeiramente incrementado para depois avaliar-se a expressão. Se estiver à direita, trata-se de uma operação de pós-incremento, em que primeiramente a expressão é avaliada, com o valor atual do operando, para só então incrementar-se o operando.

Estas regras semânticas do operador de incremento são as mesmas do operador de decremento "**--**", com a única diferença que neste último caso o operando será decrementado.

Os exemplos a seguir ilustram as operações de **pré-incremento**, **pós-incremento**, **pré-decremento** e **pós-decremento**.

Exemplo 3: Pré-incremento de uma variável do tipo int (variável "i"), e posterior atribuição de seu valor a outra variável do tipo int (variável "contador").

```
int i, contador;    // declara as duas variáveis do tipo int.

contador = ++i;     // pré-incrementa a variável i

                    // e então atribui seu valor à

                    // variável contador.
```

Exemplo 4: Atribuição do valor atual da variável "i" à variável "contador", e posterior incremento do valor da variável "i".

```
int i, contador;    // declara as duas variáveis do tipo int.

contador = i++;     // atribui o valor atual de i

                    // à variável contador e então

                    // incrementa a variável i.
```

Exemplo 5: Pré-decremento da variável "i", e posterior atribuição de seu valor à variável "contador".

```
int i, contador;    // declara as duas variáveis do tipo int.

contador = --i;     // pré-decrementa a variável i

                    // e então atribui seu valor à

                    // variável contador.
```

Exemplo 6: Atribuição do valor atual da variável "i" à variável "contador", e posterior decremento do valor da variável "i".

```
int i, contador;    // declara as duas variáveis do tipo int.
contador = i--;     // atribui o valor atual de i à variável
                    // contador e então
                    // decrementa a variável i.
```

Na seção seguinte estudaremos os operadores relacionais.

3.1. Operadores Relacionais

A linguagem Java possui operadores para o teste de equalidades e de inequalidades, representados pelas seqüências de caracteres "**==**" e "**!=**", respectivamente.

O resultado da avaliação de qualquer expressão envolvendo **operadores relacionais** é um valor booleano. Como exemplos, a condição **3==7** tem o valor **false**, enquanto que o valor de **3!=7** tem o valor **true**.

Além desses existem os operadores relacionais "<" (menor que), ">" (maior que), "<=" (menor ou igual) e ">=" (maior ou igual). Como exemplos, a expressão 5>9 retorna false, e 6<=8 retorna true.

3.2. Operadores Lógicos

Os **operadores lógicos** possibilitam a realização de operações lógicas entre os seus operandos. Eles são os seguintes: "**&&**" (função 'e' lógica), "**||**" (função 'ou' lógica) e "**!**" ('não' lógico).

Tais operadores são utilizados em diversas situações. Por exemplo, quando se quer testar se duas ou mais expressões são verdadeiras simultaneamente (função 'e'), se apenas uma de várias expressões é verdadeira (função 'ou'), ou se a negação de uma determinada expressão é verdadeira.

4. Estruturas de Fluxos de Controle

Esta última seção da aula apresenta as **estruturas de fluxos de controle**, que servem para que os programadores possam traduzir a lógica de seus algoritmos em programas.

Estas estruturas podem ser classificadas em **sentenças condicionais** e em **laços de repetição**. Ambas podem alterar o fluxo de execução de um programa, por isso o nome de estruturas de fluxo de controle.

4.1. Sentenças condicionais

A **sentença condicional** utiliza o termo reservado da linguagem "**if**" ('se') e indica que um determinado trecho do código do programa só será executado caso a expressão avaliada for verdadeira.

Por exemplo, em uma loja, um determinado prêmio pode ser concedido a um vendedor apenas na condição de o mesmo ter atingido no mês um determinado valor de vendas. Supondo que a variável "vendasMes" corresponda ao valor vendido pelo funcionário no mês, enquanto a variável "metasMes" indica o seu correspondente objetivo mensal. Então a condição de concessão do bônus ao vendedor poderá ser escrita da seguinte forma:

```
if(vendasMes > metaMes){
 performance = "Ótima";
```

```
 bonus = 500;
}
```

A sentença condicional pode ainda determinar que ações devem ser realizadas especificamente no caso de a expressão avaliada ser falsa. Neste caso, utiliza-se o termo "**else**", que significa "em caso contrário", seguido de chaves cercando as ações que devem ser realizadas nesta situação.

Tomando-se o mesmo exemplo anterior, pode-se acrescentar uma condição "**else**" para determinar as ações que devem ser realizadas caso o vendedor não atinja o objetivo mensal de vendas.

```
if(vendasMes > metaMes){
 performance = "Ótima";
 bonus = 500;
}
else{
 performance = "Satisfatória";
 bonus = 0;
}
```

Além da sentença do tipo "**if-then-else**" (outro termo usado para designar a expressão condicional que utiliza os termos if e else), existe também um outro tipo de sentença condicional, que representa uma **seleção múltipla**.

Tal construção chama-se "**switch**", e possibita a escolha de uma determinada ação (ou conjunto de ações) a serem realizados caso a variável de escolha assuma um determinado valor inteiro.

A construção "**switch**" é mais adequada do que a do tipo "**if-then-else**" quando se estiver tratando com um grande número de condições possíveis para uma determinada variável.

Por exemplo, se a variável "escolha" for do tipo inteiro, e puder assumir múltiplos valores, sendo que para cada um deles se quiser associar um conjunto de ações diferentes, podemos utilizar uma construção "switch" da forma seguinte:

```
switch(escolha){
case 1:      ...
             break;
case 2:...
             break;
default:     ...
             break;
}
```

4.2. Laços de Repetição

Os laços de repetição servem para que se possa repetir uma ou mais instruções diversas vezes. Eles podem ser do tipo determinados, quando se define quantas repetições o laço executará, ou indeterminados, quando não se tem uma definição prévia de quantas vezes o laço deverá ser executado.

4.2.1 Laços Determinados

Um **laço de repetição determinado** é criado com o auxílio do termo "**for**" (para), seguido de uma expressão entre parênteses que indica, por meio de um contador, a quantidade de interações que o laço deverá executar.

O exemplo a seguir imprime na tela os números de 0 a 10.

```
for(int i=0; i <=10; i++){

 System.out.println("Valor de i: " + i);

}
```

4.2.2. Laços Indeterminados

Um **laço de repetição indeterminado** é criado utilizando-se o termo "**while**" (enquanto) seguido de uma determinada condição booleana entre parênteses.

Esta condição booleana, enquanto for verdadeira, faz com que o laço indeterminado percorra mais uma interação (executando as instruções que estiverem no corpo da construção "while"), e ao final a condição booleana é novamente testada.

Por exemplo, se uma determinada pessoa deseja atingir um determinado valor de saldo em uma conta corrente ou de poupança, deverá acumular mensalmente uma fração de seu salário nesta conta.

O laço que permite o acúmulo do salário mensal desta pessoa poderia ser como o representado a seguir:

```
int meses = 0;

while(saldoContaCorrente < metaDeSaldo){

 saldoContaCorrente = saldoContaCorrente + fracaoPagamentoMensal;

meses++;

}

System.out.println("Número de meses: " + meses);
```

Neste exemplo, o laço "while" será executado enquanto o saldo de conta corrente for inferior ao valor de meta de saldo (que deve ser previamente definido).

Assim que o valor de saldo for igual ou superior à meta, a execução do laço while é encerrada e a próxima instrução a ser executada é a que vem depois da chave que fecha a construção do laço.

5. Conclusões

A presente aula tratou de diversos aspectos da **linguagem de programação Java**. Dentre eles, podem-se destacar: a declaração e a inicialização de variáveis, os principais

operadores e a forma de sua utilização, e também as estruturas de controle que possibilitam ao programador definir os fluxos de execução de seus programas.

Esta aula fecha um primeiro ciclo de estudos acerca da linguagem Java, o qual já permite aos alunos a escrita e o teste de diversos programas escritos nesta linguagem.

6. Referências Bibliográficas

[1] DEITEL, H.M., DEITEL, P.J. **Java – Como programar**. Terceira edição. Porto Alegre: Bookman Editora, 2001.

[2] SANTOS, Rafael. **Introdução à Programação orientada a objetos usando Java**. 8ª Reimpressão. Rio de Janeiro: Campus - Elsevier, 2003.