Programação Estruturada

Aula 02 - Tipos de dados, constantes e variáveis

Apresentação

Nesta aula, você vai conhecer e aprender a utilizar tipos de dados, constantes, variáveis e expressões típicas da linguagem Java. Todos esses assuntos são essenciais para que você realmente possa desenvolver programas em Java que manipulem dados e que realizem operações sobre eles.

Os dados de um programa geralmente necessitam ser armazenados nas chamadas variáveis do programa, as quais podem ter seus valores modificados ao longo do tempo. Como veremos nesta aula, uma variável pode ser vista como um espaço de memória do computador utilizado para armazenar um determinado tipo de dado. As variáveis são referenciadas e manipuladas a partir de seus nomes e, tanto em Java como na maioria das outras linguagens (C, Pascal, etc.), elas possuem um tipo de dado associado.

Os tipos podem ser inteiros, caracteres, ponto flutuante, dentre outros. Eles definem, por exemplo, quais são os possíveis valores que uma determinada variável pode assumir. Mas não se preocupe, você estudará todos esses conceitos que citei em maior detalhe nesta aula.

Desejo-lhe uma boa leitura!

Vídeo 01 - Apresentação

Objetivos

Depois de estudar e praticar o conteúdo desta aula, você será capaz de:

- Identificar os tipos de dados da linguagem Java e saber utilizá-los corretamente na declaração de variáveis
- Aplicar corretamente o uso de constantes nos programas.
- Criar e interpretar expressões aritméticas e boleanas na linguagem Java.

1. Tipos de dados

Durante esta seção, vamos aprender um dos principais conceitos da linguagem Java: o de tipo de dados. Tipos de dados basicamente agrupam um conjunto de possíveis valores e são utilizados, por exemplo, para declarar o tipo de uma variável, ou seja, os possíveis valores que a variável pode assumir. Usamos bastante no dia a dia alguns tipos de dados, como o tipo Inteiro, que representa o conjunto dos números inteiros, e o tipo Real, que representa o conjunto de números com componentes decimais, como no caso dos números $10,1$ e $1.127,898$. Na computação, a notação utilizada na verdade não usa separador de milhar e o separador decimal é o ponto e não a vírgula, sendo esses números representados como 10.1 e 1127.898, ok?

Conforme já mencionamos, os programas de computador geralmente precisam manipular dados, armazená-los e modificá-los. Para entender isso, comecemos estudando os tipos de dados que são utilizados na linguagem Java.

A linguagem Java possui vários tipos de dados pré-definidos, mas comumente chamados de **tipos primitivos**, como: inteiro (**int** e **long**), real (**double** e **float**), caractere (**char**), **boolean** e **void**, entre outros.

> Em Inglês, void quer dizer vazio e é isso mesmo que o void representa, um conjunto vazio de possíveis valores.
>
> Esse tipo serve para indicar, por exemplo, que uma determinada função que definimos em nosso programa não retorna nenhum valor.

Vejamos sobre esses tipos de dados em mais detalhes. Existem vários tipos numéricos relacionados aos números inteiros: byte, short, int e long. O que muda entre esses tipos é basicamente o espaço necessário na memória para armazenamento dos números representados pelos mesmos. Uma variável do tipo **byte** requer apenas 1 byte de espaço de memória, pois o tipo byte representa os

números inteiros de -128 a 127. Lembre-se de que 1 byte possui 8 bits, os quais podem ser utilizados para representar 2^8 (256) valores distintos. São eles os números negativos de -1 a -128, o 0 (zero) e os números de 1 a 127.

Os outros tipos usam uma maior quantidade de bytes da memória para representar seus números. O tipo **short** requer 2 bytes, representando valores entre - 32.768 e 32.767. Já o tipo **int** requer 4 bytes na memória, representando valores entre -2.147.483.648 e 2.147.483.647. Esse é o tipo inteiro mais utilizado na prática. Para representar números maiores, temos o tipo **long**, que usa 8 bytes para representar números entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807. Números bastante grandes, não são?

O tipo **char** representa um caractere alfa numérico (a, b, c, ..., A, B, C, ..., 0, 1, 2, ..., 9, &,*,@, etc.), requerendo 2 bytes de espaço em memória. Valores do tipo char são digitados no código fonte do programa entre aspas simples, como os caracteres 'a' e 'b'.

Já o tipo **float** representa números com precisão decimal, usando 4 bytes para isso. Existe também o tipo **double**, o qual consegue representar números decimais com precisão maior que o float, utilizando 8 bytes de espaço Em Inglês, void quer dizer vazio e é isso mesmo que o void representa, um conjunto vazio de possíveis valores. Esse tipo serve para indicar, por exemplo, que uma determinada função que definimos em nosso programa não retorna nenhum valor. em memória. O tipo **boolean**, por sua vez, representa os valores lógicos verdadeiro (true) e falso (false). Por fim, o tipo **void**, que já foi explicado anteriormente, não armazena dados, serve apenas para definir funções que não retornam valores.

Um tipo não primitivo bastante comum de se usar em Java é o tipo **String**, o qual representa um texto, e ocupa a quantidade de caracteres do texto * 2 bytes, onde esses dois bytes representa o espaço em memória para armazenar um caractere. Valores do tipo String são digitados no código fonte do programa entre aspas duplas, como nos textos "seja bem vindo!" e "até breve!".

2. Variáveis e constantes

Agora que já conhecemos os tipos de dados que são manipulados pela linguagem Java, vamos aprender a reservar espaços de memória para as variáveis. As variáveis devem ser declaradas antes de serem utilizadas e a forma geral de se fazer isso é a seguinte:

tipo_da_variável lista_de_variáveis;

Note que após apresentar o tipo de uma variável, podemos declarar diversas variáveis ao mesmo tempo. Por exemplo, são declarações válidas de variáveis:

```
1 double preco;
2    char inicial, letra;
3 double altura, peso, envergadura;
```

Você também pode declarar uma variável e já inicializá-la, como nos exemplos a seguir:

```
1 int idade = 16;
2 double altura = 1.75, peso = 75.4;
```

Em Java, a declaração das variáveis pode ocorrer em diferentes lugares dentro do código. A forma e o local de declaração de uma variável determina o escopo da variável, ou seja, o trecho de código na qual a variável é válida. Uma variável é chamada de global se ela pode ser acessada em qualquer parte de seu sistema. Como um sistema em Java possui várias classes (módulos que contêm declarações de variáveis e rotinas), isso quer dizer que uma variável declarada em uma classe como global pode ser acessada por qualquer outra classe do seu sistema.

Caso a variável declarada não seja global, ela será local a um determinado escopo, ou seja, poderá ser manipulada por apenas parte do código da classe na qual ela foi declarada. Para melhor exemplificar os conceitos apresentados, vamos examinar o seguinte trecho de código.

```
1  public class Programa {
2     public static int varGlobal = 5;
3     private static int varLocalClasse = 6;
4
5     public static void main(String[] args) {
6     int varLocalRotina = 10;
7
8     if (varGlobal == 0) {
9     int varLocalComando = varLocalRotina;
10    System.out.println(varLocalComando);
11 }
12 System.out.println(varLocalRotina);
13 // System.out.println(varLocalComando);
14 System.out.println(varGlobal);
15 System.out.println(varLocalClasse);
16 }
17
18 public static void rotina2() {
19 // System.out.println(varLocalRotina);
20 // System.out.println(varLocalComando);
21 System.out.println(varGlobal);
22 System.out.println(varLocalClasse);
23 }
24 }
```

A variável **varGlobal**, declarada na linha 2, é considerada uma variável global e é acessível em qualquer parte do programa. Isso porque ela foi declarada usando-se as palavras-chave da linguagem **public static**. Essa forma de declaração não é aconselhada, visto que variáveis globais podem dificultar a manutenção dos seus programas. Observe que se a variável é global, ela pode ser alterada por qualquer parte do programa. Caso você precise alterar uma rotina que modifica o valor de uma variável global, precisará olhar todo o restante do código do seu sistema para ter certeza de que sua alteração não irá modificar erroneamente o comportamento de alguma outra rotina do programa que também utilize aquela variável global.

Já a variável **varLocalClasse**, declarada na linha 03, é uma variável local à classe (chamada em POO como variável de instância), ou seja, seu escopo é o corpo da classe. Isso porque ela foi declarada fora de qualquer uma das rotinas declaradas na classe Programa (rotinas main e rotina2). Note que fazemos o uso de **private static** ao invés de **public static** ao declararmos variáveis locais a uma classe. Observe também que tanto o seu valor como o valor da variável global varGlobal estão sendo impressos nas rotinas main e rotina2 (ver linhas 13 e 14, e 20 e 21).

Temos também uma variável chamada de **varLocalRotina** que foi declarada no início da rotina main (linha 06). Variáveis declaradas dentro de rotinas não fazem uso das palavras-chaves **public static**, ok? Você entenderá isso melhor na disciplina de Programação Orientada a Objetos. O escopo de varLocalRotina vai do próximo comando (linha 07) até o último comando da rotina (linha 15). Variáveis declaradas como parâmetros de rotinas (funções ou procedimentos) possuem escopo similar: todo o corpo da rotina.

Temos também a variável **varLocalComando** (linha 08), que foi declarada dentro do bloco de código do comando *if* da linha 07. Isso quer dizer que o escopo dessa variável vai da linha 07 até a última linha do bloco do *if* (linha 10). Note que não tem como se ter acesso a essa variável na linha 12, que fica fora do bloco do *if*. Também não dá para acessar as variáveis varLocalRotina e varLocalComando a partir das linhas 18 e 19, pois o escopo dessas variáveis não se estendem a códigos de outras rotinas. Se forem removidos os comentados desses comandos, eles irão gerar um erro de compilação.

Vídeo 02 - Variáveis Globais e Locais

Também é importante esclarecer o conceito de constantes. Constantes são valores fixos que não podem ser alterados durante a execução do programa. Para declarar uma constante em Java, adicionamos a palavra final a essa declaração. Veja como as variáveis mostradas poderiam ser declaradas como constantes:

```
1 public static final int varGlobal = 5;
2    private static final int varLocalClasse = 6;
3    final int varLocalRotina = 10;
4 final int varLocalComando = varLocalRotina;
```

Basicamente, o que muda é o uso da palavra-chave final usada na declaração para indicar que o nome indicado é uma constante e não uma variável. Para definir o nome das variáveis e constantes, você deve seguir certas regras que garantem a legibilidade e corretude do código. Algumas dessas regras devem realmente ser seguidas, pois se assim não for feito, o compilador indicará um erro. São regras recomendadas:

1. O primeiro caractere deve ser uma letra, um _ (*underscore*) ou o símbolo $;
2. O restante do nome deve ser formado pelos seguintes caracteres: letras, números, *underscore* ou $;
3. não se deve usar espaços em branco nos nomes, mas usar os nomes juntos, colocando a primeira letra de cada um dos nomes seguintes em maiúsculo, como alturaAluno para representar a altura de um certo aluno;
4. não se deve utilizar acentos nem cedilha.

Vídeo 03 - Constantes

Atividade 01

1. Identifique os tipos dos valores abaixo. Use para isso os tipos int, float, char e String:
    1. 1000
    2. "09"
    3. -1.56
    4. "VERDADE"
    5. -456
    6. 34
    7. 'C'
    8. 45.8976
    9. "BRASIL"
    10. 'l'
    11. -5.6

12. 300

2. Abaixo, temos exemplos de identificadores. Assinale os identificadores válidos e descreva o erro dos operadores inválidos.

    1. endereco
    2. 21brasil
    3. fone$com
    4. nome_usuario
    5. usuario6
    6. nome*usuario
    7. end*a-6
    8. #media
    9. nome aluno
    10. média
    11. _pais
    12. MediaTurma

3. Operador de atribuição (=)

O comando de atribuição de valor a uma variável é realizada pelo operador “=”. Assim, o operador de atribuição nos possibilita atribuir um dado valor a um espaço de memória previamente declarado (variável). É importante que o dado a ser armazenado seja compatível com o tipo da variável que receberá a atribuição. Por exemplo, as variáveis reais podem receber valores reais e inteiros, mas não boleanos ou textos.

No entanto, uma variável inteira não pode receber um valor real, pois não haverá como armazenar a parte decimal desse valor. Veja a seguir a sintaxe geral do operador de atribuição representado pelo símbolo =.

variável = valor;

No lado esquerdo do operador da atribuição, temos a variável que vai receber o valor. No lado direito do operador, temos o valor que será atribuído à variável. Ao final da linha de atribuição, você deve colocar um ponto e vírgula (“;”) para indicar o fim do comando. E você pode fazer atribuições diretamente na declaração das variáveis, como já havíamos visto, ou posteriormente, como no exemplo a seguir:

varLocalComando = 10;

4. Valores padrão (=)

Ao declarar variáveis, podemos ou não inicializá-las com valores iniciais. Veja os seguintes exemplos:

```
1 int x = 10;
2 int y;
```

Qual o valor de x e qual o valor de y? Sabemos que o valor da variável x é 10, mas e quanto a y? No caso da maioria das linguagens, a declaração de variáveis sem inicialização de valores (caso da variável y do exemplo) implica na inicialização automática de um valor padrão. Esse valor padrão pode variar de linguagem para linguagem, mas, na maioria dos casos, seguirá os valores apresentados na tabela a seguir.

Tipo de dado	Valor padrão
Tipos inteiros (byte, short, int, long, etc.)	0
Tipos numéricos com casas decimais (float, double, etc.)	0.0
Caractere	'\u0000'
String (ou qualquer objeto)	null

Tipo de dado	Valor padrão
Boleano	false

Tabela 1 - Valores padrões dos principais tipos de dados suportados em Java.

Note que variáveis numéricas são inicializadas com o valor zero, variáveis boleanas com o valor *false* e variáveis do tipo caractere com um caractere especial denotado pelo código \u0000 (padrão de codificação Unicode). Já o tipo String, com um valor especial chamado de null (você irá aprender melhor sobre isso em orientação a objetos).

Porém, em Java, algumas variáveis não podem deixar de serem utilizadas. Veja como exemplo o código a seguir:

```
1  public class Programa2 {
2
3      private static int varLocalClasse;
4
5      public static void main(String[] args) {
6          int varLocalRotina;
7          System.out.println(varLocalClasse);
8          System.out.println(varLocalRotina);
9      }
10 }
```

Se você tentar compilar esse código, verá que a linha 08 irá gerar um erro de compilação. O compilador não deixa que uma variável local a uma rotina seja utilizada sem ser inicializada diretamente pelo programador. Isso é feito para se evitar erros. É bastante comum um programador criar uma rotina, declarar algumas variáveis nela e esquecer-se de inicializá-las. Essa regra não se aplica, porém, a variáveis declaradas fora de rotinas, dentro das classes (variáveis globais, por exemplo).

5. Expressões aritméticas e boleanas

Alguns operadores em Java possuem mais de uma função. O operador + pode ser utilizado para concatenação de valores literais, resultando em um texto, por exemplo. Agora veremos o uso do + e de outros operados para realização de operações aritméticas. Em particular, veremos os seguintes operadores aritméticos:

```
1 + soma
2 - subtração
3 * multiplicação
4 / divisão
5 % resto da divisão
```

Esses operadores, todos binários, são utilizados da mesma forma que aprendemos na matemática. Talvez apenas o operador resto da divisão (%) seja novo para você. Basicamente, a operação **a % b** retorna o quanto sobra de uma divisão inteira de a por b, se ela não for exata. Por exemplo, 7 % 2 é igual a 1. Isso porque a divisão inteira de 7 por 2 é igual a 3. Para ver quanto sobra, é só multiplicar 2 (divisor) x 3 (divisão inteira), que dá 6. Se subtrairmos 7 por 6, vemos o que sobrou, que foi o valor 1.

Além disso, é preciso observar a precedência das operações. Qual é o valor resultante da expressão 3 - 2 * 5? Primeiro, temos que calcular as operações multiplicativas (*, /, %) para depois calcular as operações aditivas (+, -). Parênteses, porém, podem ser utilizados para forçar o cálculo de uma expressão em uma determinada ordem. Exemplos:

```
1 int a = 3 - 2 * 5;  // calcula-se primeiro 2*5
2 int b = (3 - 2) * 5; // calcula-se primeiro 3-2
```

O valor da variável a é igual a -7, enquanto o valor de b é igual a 5!

Além das expressões aritméticas, temos também as expressões boleanas, as quais são verdadeiro (**true**) se elas forem atendidas, ou falso (**false**) caso contrário. Para montar tais expressões, você fará uso geralmente de operadores relacionais de

>, >=, <, <=, == e !=, sendo **==** o operador de igualdade, e **!=** o de diferença. Veja exemplos de expressões que são verdadeiras, considerando os valores de a e b mostrados anteriormente:

```
a == -7
b != -7
b >= 0
```

Por outro lado, são exemplos de expressões falsas:

```
a == 8
b != 5
a > 0
b < a
```

Você pode compor essas expressões mais simples através dos operadores lógicos vistos em Lógica de Programação: operador NÃO (!), operador E (&&) e o operador OU (||). Veja exemplos a seguir de expressões verdadeiras:

```
a == -7 && b == 5
!(a == 5)
a == -7 || a == 0
```

Vídeo 04 - Precedência de Operadores

6. Leitura Complementar

Para mais detalhes sobre os tipos de dados primitivos suportados pela linguagem Java, leia o seguinte conteúdo:

<http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

7. Resumo

Nesta aula, você estudou a definição dos tipos das variáveis, bem como expressões, tanto as constantes, como as que utilizam esses dados para manipular os dados do programa. Esse assunto é de grande importância, pois a partir dele você será capaz de implementar seus primeiros programas, de declarar e inicializar as variáveis.

8. Autoavaliação

1. Construa um programa que utilize uma variável inteira global e atribua o valor 20 a ela. Declare outras 5 variáveis inteiras locais ao programa principal e atribua os valores 30, 40, ..., 60 a elas. Declare 4 variáveis caracteres e atribua a elas as letras g,a,t,o. O programa deverá imprimir, usando todas as variáveis declaradas: as variáveis inteiras que contêm os números: 20,30,40,50,60 e as variáveis caracteres que contêm as letras citadas acima.

2. De acordo com as declarações a seguir, assinale os comandos de atribuição inválidos e descreva o erro.

```
1 int NUMERO, X, SOMA;
2 float MEDIA, K, L;
```

( ) SOMA = NUMERO + 2;
( ) MEDIA = SOMA;
( ) NUMERO = K + L;
( ) X = X + 1

( ) L = SOMA - K;
( ) SOMA + 2 = NUMERO + 10;
( ) S = SOMA;
( ) X = SOMA - NUMERO;

3. Declare e inicialize variáveis do tipo String que formam o seu nome completo. Depois, imprima cada uma delas na tela.

4. Construa um programa que mostre na tela o seguinte resultado:
   um
   dois
   três

5. Usando uma variável contadora de linhas, faça uso do valor dela para mostrar o texto abaixo:
   Esta é a linha 1.
   Esta é a linha 2.

9. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.