DR1

Disciplina Regular 1 Fundamentos do Desenvolvimento Java

Graduação em Engenharia de Software - 2020

DR1

Etapa 1 Aula 2

Variáveis e Operadores

DR1

Declaração de Variáveis

Identificadores

DR1

- Usados para identificar variáveis, constantes, classes e métodos.
- Existem regras de formação de identificadores e algumas convenções que são seguidas por todos os que programam em Java.
- As regras são forçadas pelo compilador.
- As convenções devem ser seguidas para um melhor entendimento dos códigos.

Identificadores

DR1

- Começa com letras, _(sublinhado) ou $ - esses dois últimos não são usuais.
- Pode conter dígitos no meio ou no final.
- As palavras reservadas não podem ser usadas.
- Quando usa maiúscula?
  - Primeira letra do nome da classe.
  - Separador de palavras.
  - Constantes são escritas com todas em maiúsculo.
  - O resto todo é em minúsculas.

Literais

- Literais inteiros
  - Formato implícito (int) → 7, 3, 1970
  - Formato explícito (long) → 6000000000L, 16000000l
  - Formato octal → 04, 037, 07
  - Formato Hexadecimal → 0xA9, 0X58, 0x20

- Literais de ponto flutuante
  - Formato implícito (double) → 2.0952
  - Formato explícito (double) → 2.17d, 2.3D
  - Formato explícito (float) → 1.75f, 2.10F
  - Formato científico (double) → 7.2e-11, -18.911E21
  - Formato científico (float) → 1.12E3f, 7.9e-8F

Tipos Primitivos

DR1

byte	(8 bits)	Numéricos sem casa decimal
short	(16 bits)	
int	(32 bits)	
long	(64 bits)	
float	(32 bits)	Numéricos com casa decimal
double	(64 bits)	
char	(16 bits)	Caracter da tabela unicode
boolean	(JVM)	true/false

Tipos Primitivos

DR1

- Inteiros – byte, short, int e long.

- A diferença entre eles está no intervalo de valores que cada um pode suportar:

```
byte menor = 10; // 1 byte
short pequeno = 456; // 2 bytes
int normal = 10252; // 4 bytes
long muitoGrande = 6263732239L; // 8 bytes
```

Tipos Primitivos

DR1

- Ponto flutuante – float e double
  - float - precisão simples (sete dígitos) que utiliza 32 bits de armazenamento.
  - Tornam-se imprecisas para valores muito grandes ou muito pequenos.
  - Úteis quando precisamos de um valor fracional, sem grande necessidade de precisão.
  - double - precisão dupla (15 dígitos) que utiliza 64 bits de armazenamento.

```
float numeroReal = 10.9f;// 4 bytes
double numero = 6745.9E13;// 8 bytes
```

Tipos Primitivos

DR1

- Textual – char - 16 bits - 2 bytes.

```
char meuCaractere = 'L';
char meuCharUnicode = '\u0058';
```

- A contrabarra indica uma sequência de escape.

Tipos Primitivos

DR1

	‘\b’	→ backspace
⚠	‘\t’	→ tab
	‘\f’	→ form feed
⚠	‘\n’	→ line feed
	‘\r’	→ carriage return
	‘\’’	→ aspas simples
	‘\”’	→ aspas duplas
	‘\\’	→ contrabarra

Tipos Primitivos

DR1

- Lógico – boolean.

```
boolean status = true;
boolean continuar = false;
```

- Os literais do tipo boolean são escritos em letra minúscula.

Variáveis

DR1

- Java é uma linguagem “fortemente tipada”
- Toda variável deve ser declarada explicitamente antes de seu uso.

```
int populacaoMundial;
char opcao;
float cotacaoDolar, preco;
boolean status = true;
double produtoInternoBruto;
```

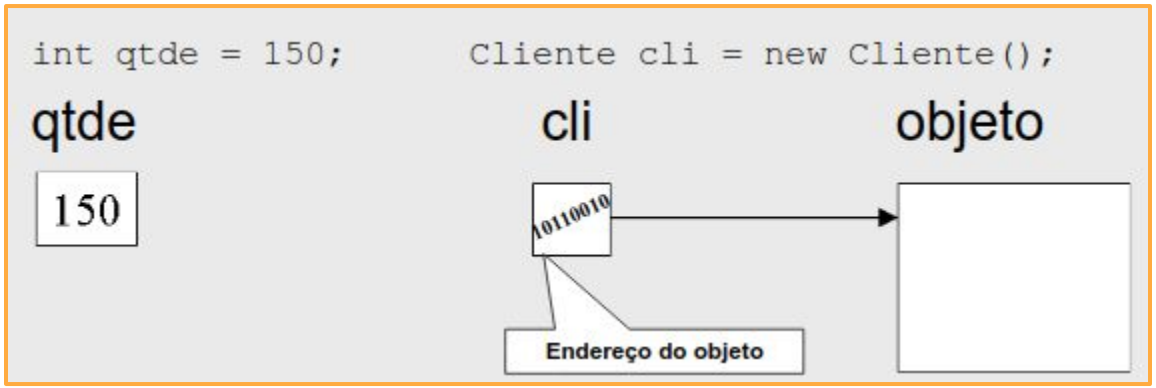
Uma dessas declarações está errada.

O erro se deve ao desconhecimento dos tipos de variáveis de Java.

Qual declaração está conceitualmente errada?

Valor e Referência

- Em Java existem dois tipos de variáveis:
  - Valor: variáveis de tipos primitivos.
  - Referência: variáveis de classes – “apontam” para objetos em memória.



Processamento de Expressões

DR1

- Em Java, quando uma expressão aritmética contém vários tipos de dados, é realizada uma conversão interna de todos os componentes para o maior tipo que estiver presente na expressão.

```
numDouble + numInt + numFloat + numLong
```

Nessa expressão todos os operandos serão ajustados para **double** a fim de permitir obter um resultado

- Se um dos operandos for uma string, todos serão ajustados para String . Desta forma, todos os operandos serão concatenados (só funciona com soma).

Type Cast

DR1

- Java não opera quando um tipo não “cabe” no outro. É necessário fazer um “ajuste” explícito:

```
public class TiposPrimitivos{
   public static void main(String args[]){
     int a = 5, b = 2;
     int c;
     c = a / b;
     System.out.println("c = " + c);

     double d;
     d = a / b;
     System.out.println("d = " + d);

     double e;
     e = (double) a / b;
     System.out.println("e = " + e);
   }
}
```

$c = 2$

$d = 2.0$

$e = 2.5$

Type Cast

```
long grande = 890L;// 64 bits
int pequeno = (int) grande; //truncou?
char letra = (char) 87; //letra 'W'
```

- Sempre que possível é feito o “ajuste” implícito.

- Alguns “ajustes” implícitos permitidos:

byte > short > int > long > float > double

DR1

Declaração de Métodos

Estrutura de um Método

DR1

- Métodos têm a seguinte estrutura genérica:

```
<visibilidade> <static> <tipo_retorno> <nome_metodo> (<parâmetros>) {

    //corpo ou escopo do método. Fica entre chaves { }.
}
```

<visibilidade> - public, private, <package>, protected;

<static> - com a palavra static ou sem ela;

<tipo_retorno> - void, tipo primitivo ou classe;

<nome_método> - valem as convenções;

<parâmetros> - lista de variáveis a serem preenchidas na chamada do método.

Estrutura de um Método

DR1

- Métodos retornam no máximo um valor.
- Se [tipo_retorno] não for void, o corpo do método tem que ter um return.
- Um método não pode ser definido dentro de outro método.
- Métodos devem ter sua função bem definida para promover a reutilização de código.
- Métodos não devem ter muitas linhas de código, nem muitas variáveis.
- Teste SEMPRE os parâmetros recebidos antes de executar o código do método.

Chamadas a Métodos

DR1

- Dentro da definição da própria classe:

```
nomeDoMetodo();
```

- Métodos de classe – são estáticos:

```
int x = Integer.parseInt("12345");
double d = Math.random();
```

- Métodos de instância – são dinâmicos:

```
ContaCorrente cc = new ContaCorrente();
cc.consultarSaldo();
Cliente cli = new Cliente();
cli.getNome();
```

Visibilidade

Modificadores	Mesma classe	Mesmo package	Subclasses	Qualquer lugar
private	x			
<package> *	x	x		
protected	x	x	x	
public	x	x	x	x

package xyz

```
class Teste1 {
  private int x;
  int y;//*
  protected int z;
  public int w;
  metodo m1( ){
    x,y,z,w -> ok
  }
}
class Teste2
  x -> não
  y -> ok
  z -> ok
  w -> ok
```

package abc

```
class Teste3
   x -> não
   y -> não
   z -> não
   w -> ok

class Teste4 extends Teste1
   x -> não
   y -> não
   z -> ok
   w -> ok
```

Exercício

DR1

- Crie uma classe TestaMetodos que contenha os seguintes cabeçalhos de métodos:
  - Método público e dinâmico para calcular a soma de dois números reais.
  - Método privado e estático que inverta a ordem dos caracteres de uma frase.
  - Método de pacote que verifique se um servidor está ativo ou não.
  - Método público e estático que verifique se um CNPJ é válido.

DR1

Operadores

Aritméticos

DR1

- Requerem dois operandos.
- Operações aritméticas básicas.
- Funcionam com variáveis e literais.

```
int x, y, z;

x = 229 + 23;// adição
y = 73 - 9;  // subtração
z = 72 * 6;  // multiplicação
x = y / z;   // divisão
y = x % z;   // resto da divisão
```

Aritméticos

DR1

- O operador ++ incrementa de 1.
- O operador -- decrementa de 1.
- Duas formas de utilização: pré-fixada e pós-fixada.

```
int x, y;
x = 2;
y = x++; // pós-fixado => x = 3 e y = 2

x = 2;
y = ++x; // pré-fixado => x = 3 e y = 3
```

Aritméticos

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
```

Relacionais

DR1

- Os operadores relacionais sempre retornam um valor do tipo boolean:

>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Lógicos

DR1

- Os operadores lógicos mais usados são:

&&	E
||	Ou
!	Não

- Estes operadores são chamados de curtos pois avaliam apenas o que for necessário para retornar um resultado.

- Existem operadores longos que avaliam toda a expressão:

&	E (longo)
|	Ou (longo)
^	Ou exclusivo

Lógicos

```
if(x > y & y < z)
```

V	testa
F	testa

```
if(x > y && y < z)
```

V	testa
F	não testa

```
if(x > y | y < z)
```

V	testa
F	testa

```
if(x > y || y < z)
```

V	não testa
F	testa

```
if(x > y ^ y < z)
```

V	F
F	V