Disciplina: LPOO
Prof. Anderson Viçoso de Araujo

Aula 02: Variáveis, Tipos de dados e Operadores

andvicoso@facom.ufms.br
http://prof.facom.ufms.br/~andvicoso/

Variáveis

- Uma **variável** representa uma informação armazenada pelo programa
- Após a **declaração de uma variável** um espaço de memória é reservado para o **armazenamento de informações**
  - As **informações** contidas em uma variável **podem mudar** durante a execução do programa (por isso esse nome ☺)
- Cada variável tem um **nome** (identificador), um **tipo** e um **valor**
- Podem ser de diferentes tipos
  - Em Java podemos criar variáveis de **tipos primitivos** ou de **objetos**
- Podem ser declaradas em **qualquer posição** no código
  - Diferentemente de outras linguagens que obrigam que a declaração ocorra no início do código

Variáveis – Nome/Identificador

- Servem para identificar uma variável
- **Não é possível ter duas variáveis com o mesmo nome** no mesmo escopo*
- Regras:
  - Não podem ser palavras reservadas da linguagem
  - Não podem conter espaços e caracteres especiais , a não ser **$** e _
  - Não podem começar com números
- Em geral, nomes das variáveis começam com letras minúsculas e cada primeira letra de cada palavra subsequente esteja em maiúscula
  - **Não é obrigatório**, apenas uma recomendação
- Exemplos:
  - Válidos: `a_b, x3, $posicao, _maiorValor ...`
  - Inválidos: `ab 3, 2x, for, if, *loop ...`

Variáveis - Sintaxe

- Sem inicialização:
  - `[características] tipo nome;`
- Com inicialização:
  - `[características] tipo nome = valor_inicial;`
  - Serve para atribuir um valor inicial à variável
- Podem ser declaradas mais de uma variável do mesmo tipo na mesma linha, mas não é comum em Java
  - `tipo nome, nome2;`
  - `tipo nome = valor1, nome2 = valor2;`

Variáveis - Tipos

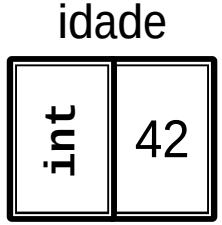
- O tipo pode ser:
  - Tipos primitivos: **boolean, char, byte, short, int, long, float** e **double**
  - Objetos*: Pessoa, String, Date, User, ...

*Vamos falar deles mais tarde...

Tipos Primitivos

- É a representação de uma variável de tipos básicos de uma linguagem
- Um inteiro, um número de ponto flutuante, um booleano ou um caractere:

```
char b = 'b';
int idade = 42;
boolean h = true;
double xyz = -22.555;
```



Resumo

Tipo	Descrição
boolean	Pode assumir o valor **true** ou o valor **false**
char	Caractere em notação Unicode de **16 bits**. Serve para a armazenagem de dados alfanuméricos. Também pode ser usado como um dado inteiro com valores na faixa entre **0 e 65535**.
byte	Inteiro de **8 bits** em notação de complemento de dois. Pode assumir valores entre **-2^7=-128 e 2^7-1=127**.
short	Inteiro de **16 bits** em notação de complemento de dois. Os valores possíveis cobrem a faixa de **-2^{-15}=-32.768 a 2^{15}-1=32.767**
int	Inteiro de **32 bits** em notação de complemento de dois. Pode assumir valores entre **-2^{31}=2.147.483.648 e 2^{31}-1=2.147.483.647.**
long	Inteiro de **64 bits** em notação de complemento de dois. Pode assumir valores entre **-2^{63} e 2^{63}-1.**
float	Representa números em notação de ponto flutuante normalizada em precisão simples de **32 bits** em conformidade com a norma IEEE 754-1985. O menor valor positivo representável por esse tipo é **$1.40239846e^{-46}$ e o maior é $3.40282347e^{+38}$**
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de **64 bits** em conformidade com a norma IEEE 754-1985. O menor valor positivo representável é **$4.94065645841246544e^{-324}$ e o maior é $1.7976931348623157e^{+308}$**

Inteiros

- Decimal (base 10):

```
int size = 374;
int a = 0;
int totalLoss = -1234;
```

- Octal (base 8)
  - Somente dígitos de 0 a 7
  - É representado um zero na frente do número

```
int seis = 06;//6 decimal
int sete = 07;//7 decimal
int oito = 010;//8 decimal
int nove = 011;//9 decimal
```

- Hexadecimal (base 16)
  - **0 1 2 3 4 5 6 7 8 9 a b c d e f**
  - Java aceita letras maiúsculas ou minúsculas para `abcdef`

```
int x = 0X0001;
int y = 0x7fffffff;
int z = 0xDeadCafe;

Valores de x, y e z?
```

Inteiros e Inteiros Longos

- Todos esses inteiros (decimal, octal e hexa), são definidos como **int** por padrão, mas também podem ser especificados como **long**:

```
long jo = 110599L;
long so = 0xFFFF1;
```

- Inteiros do tipo **long** têm 64 bits

Números de Ponto Flutuante

- São definidos como:
  - **double** (64 bits) é o **padrão**
  - **float** (32 bits)
    - Para atribuir um número de ponto flutuante a uma variável **float** é necessário anexar o sufixo **F** ou **f** ao final do número

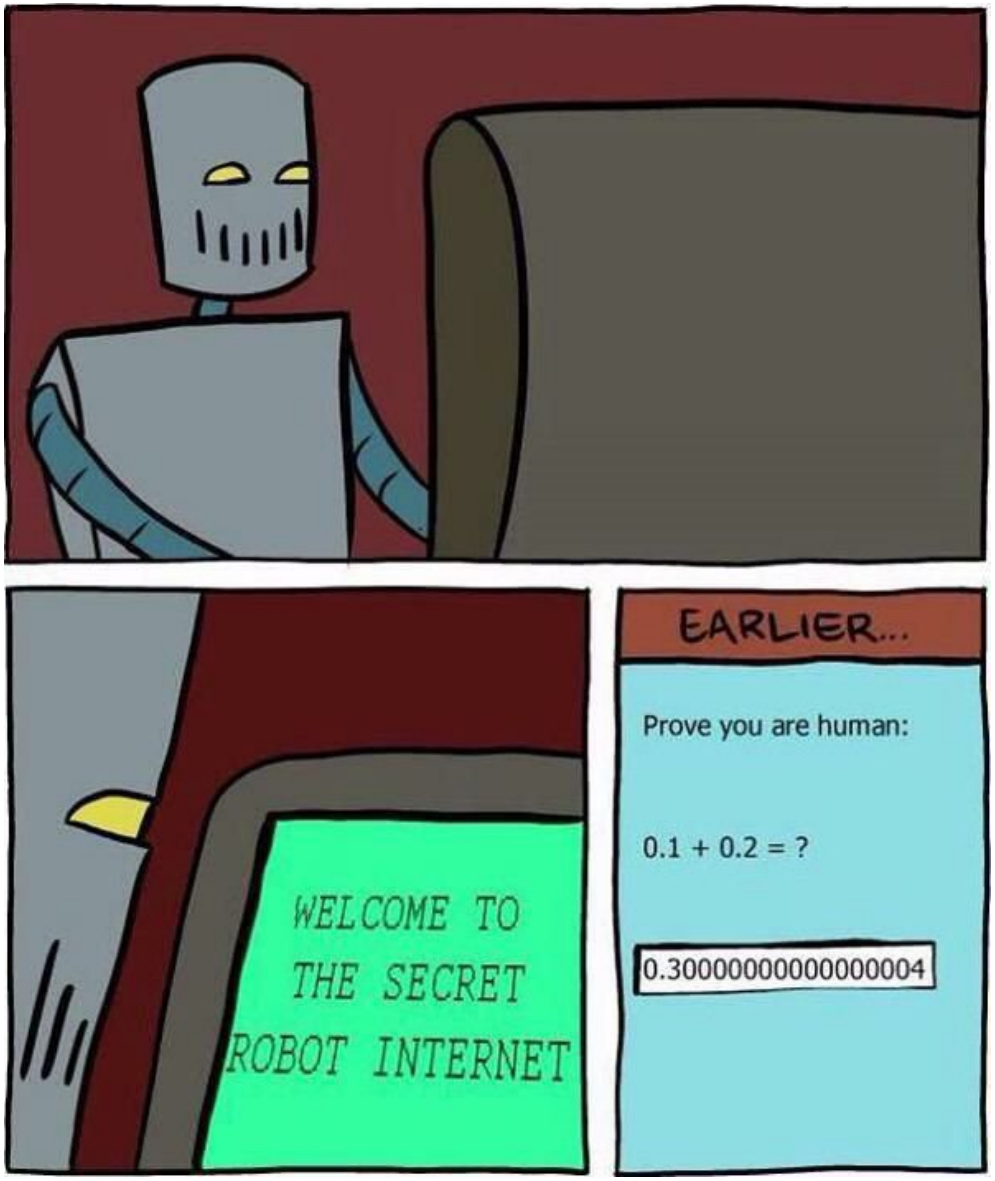
```
double d = 323123121.32323232;

float f = 23.332323233; // Erro ao compilar, possível perda de precisão
float g = 343423423.432423423F; // OK, possui o sufixo "F"
```

  - Opcionalmente, você também pode anexar **D** ou **d** aos valores **double**

```
double e = 3432434.4324234D;
```

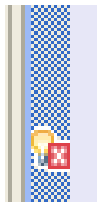
Curiosidade...

- O que sai na tela ao executarmos a linha:
  - `System.out.println(.1 + .2 == .3);` → `false`
  - `.1 + .2 == 0.30000000000000004`
- Isso acontece pois a representação dos números decimais é realizada através de potências de 2^{-n}
  - `0.1` → `0.10000000149011612`
    - Binário: `00111101110011001100110011001101`
    - $2^{-1}*0+2^{-2}*0+2^{-3}*1+...+2^{-n} = 0.5+0.25+0.125+...$
  - `0.2` → `0.20000000298023224`
- Da mesma forma, podemos representar o número 1/3 em decimal?
  - 0.333333333...
- http://www.h-schmidt.net/FloatConverter/IEEE754.html


WELCOME TO
THE SECRET
ROBOT INTERNET
EARLIER...
Prove you are human:
0.1 + 0.2 = ?
0.30000000000000004

Booleanos

- Um valor **boolean** só pode ser definido como **true** ou **false**
- **Não podemos usar outros valores**, como por exemplo `0` para `false` e `1` para `true`



```
boolean t = true; // Válido

boolean w = 0; //Erro de compilador
```

Caracteres

- Representados por um único caractere entre aspas simples

```
char n = 'a';
```

- Os caracteres são simplesmente inteiros de 16 bits sem sinal
- Isso significa que pode-se atribuir um literal numérico que esteja no intervalo de 16 bits (65535 ou menor)

```
char r = 0x345;
char s = 70000; // A conversão é necessária. 70000 está fora do intervalo char

char s = (char) 70000;
```

char x int

- Tabela ASCII
  - É uma forma universal de representação do caracteres por meio de números
- Para saber o número ASCII de um char é só copiar ele para um inteiro:

```
int x = 'a';
sysout(x);
```

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Strings

- Para representar uma cadeia de caracteres (**string**):

```
String s = "Bill Joy";
```

- Strings em Java são **objetos especiais**
  - Existe um tratamento e gerenciamento específico para este tipo de variável

Operador de Atribuição

- O valor é inserido à direita do sinal '='

```
int i = 5;// i recebe o valor 5
int j = i; // j recebe uma CÓPIA do valor de i
i = i + 1;// i vira 6 e j continua 5
```

- Para variáveis de tipos primitivos, o valor é **copiado** para a variável
- É possível atribuir um valor a mais de uma variável ao mesmo tempo, mas não é comum

```
int i = 5;// i recebe o valor 5
int j = 4; // j recebe o valor 4

i = j = 10;// ambos recebem o valor 10;
```

Conversão de Tipos Primitivos (*cast* ou *typecast*)

- A conversão permite que valores primitivos sejam transformados de um tipo para outro
- As conversões podem ser:
  - **Implícita**: Conversão ocorre automaticamente

```
int a = 100;
long b = a; // Conversão implícita, um valor int sempre cabe num long
```

  - **Explícita**: Você informa o compilador que reconhece o perigo e aceita a responsabilidade de perder precisão

```
float a = 100.001f;
int b = (int)a; // Conversão explícita, um tipo de float pode perder
                // informações como um tipo de int
```

Exercício Prático

1. Crie um código Java (pode ser no papel) com a definição das variáveis para:
   - Sua idade
   - Seu nome
   - Seu peso
   - Tipo Sanguíneo (sem fator RH - +)
   - Seu aniversário
   - Número que calça
   - Casado ou não
2. Quais são as definições de variável inteira em hexadecimal para os valores `-1` e `-2147483648`?

Operadores

Operadores

- Operadores são **símbolos especiais** que controlam como uma expressão deve ser avaliada
- Podem ser de diferentes tipos:
  - Aritméticos
  - Atribuição (com atalhos)
  - Comparação
  - Igualdade
  - Concatenação de strings
  - Etc ...

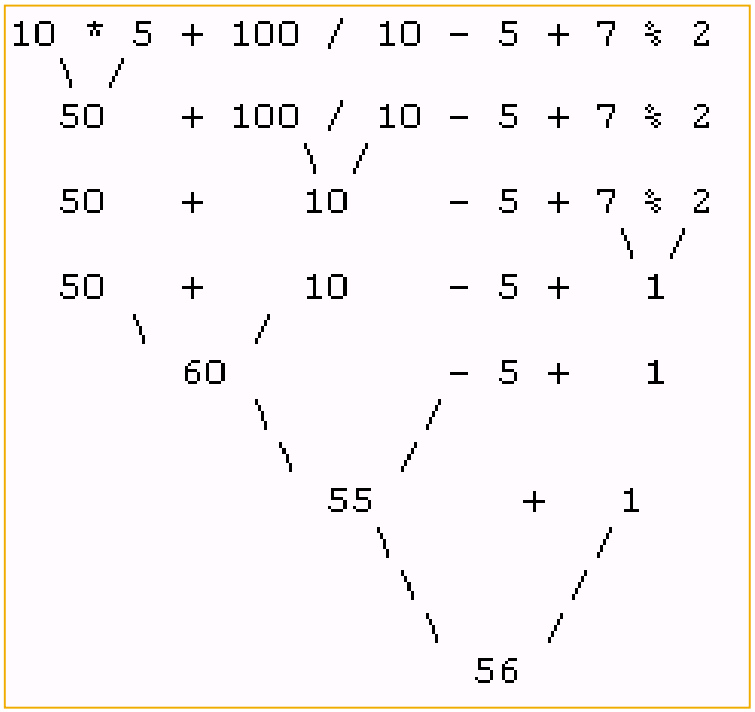
Operadores Aritméticos

- Operadores aritméticos funcionam de forma similar à matemática
  - Soma (+), Subtração (-), Divisão (/) e Multiplicação (*)
  - O resto da divisão entre dois números pode ser obtido através do operador (%)

```
int x = 15;
int y = x % 4;

System.out.print("Resto: ");
System.out.print(y);
```

Operadores Aritméticos

- As expressões, em geral, são avaliadas da **esquerda para a direita**
- Parênteses e alguns operadores podem ter maior **precedência**
  - Operadores "*", "/", "%" têm maior precedência do que "+" e "-“, por exemplo



Operadores de Atribuição Compostos

O operador de atribuição (**=**) é usado para armazenar valores em variáveis, mas existem variações dele (atalhos)

=

+=	-=	*=	/=	%=

Operação	Atalho
x = x + 5	x += 5
x = x – 5	x -= 5
x = x * 5	x *= 5
x = x / 5	x /= 5
x = x % 5	x %= 5

Operadores de Comparação

- Java tem 4 operadores de comparação:
  >, >= , <, e <=
- Sempre resultam em um valor booleano (**true** ou **false**)
- Utilizados em testes de comparação (testes lógicos)
- Podem ser usados apenas para testar tipos numéricos

```
int x = 8;
if(x < 9){
      //Faz algo
}
```

```
boolean b = 8 > 3;
```

Operadores de Igualdade

- Há dois operadores de igualdade:
  - == e !=
- Podem ser testados em:
  - Números, Caracteres, Booleanos
  - Variáveis de Referências (Objetos*)
    - Para estes tipos em especial vamos ver uma outra maneira de verificar a igualdade...
- Igualdade de tipos primitivos
  - Exemplos:
    - `‘a’ == ‘a’`
    - `‘a’ == ‘b’`
    - `5 != 6`
    - `(true == false)`
    - `boolean cincoEhDiferenteDeQuatro = 5 != 4;`

Operador de Concatenação de Strings

- O símbolo “+” também pode ser usado para **concatenar Strings**
- Regras:
  - Se um dos operandos for uma String, o resultado será a concatenação dos operandos
  - Se os dois operandos forem numéricos, o resultado será a soma dos operandos

```
String a = "String";
int b = 3;
int c = 7;
System.out.println(a + (b + c));
```

```
int d = 2;
System.out.println("" + d + 3);
```

Operadores Acréscimo e Decréscimo

- Java possui dois operadores que aumentam ou diminuem o valor de uma variável numérica em exatamente uma unidade:
  - ++ acréscimo (prefixo)
  - -- decréscimo (prefixo)
  - acréscimo++ (sufixo)
  - decréscimo-- (sufixo)
- A partir daqui é interessante fazer uso da ferramenta:
  - http://cscircles.cemc.uwaterloo.ca/java_visualize/

DICA - Operadores Acréscimo e Decréscimo

- Se for **prefixo**, a operação de acréscimo ou decréscimo deverá ser realizada **antes** de sua utilização
- Caso seja **sufixo**, deve ser utilizado e **depois** a variável deve sofrer a alteração de valor

```
int teste = 0;
System.out.println("Resultado: " + teste++);
System.out.println(teste);
System.out.println(++teste);
System.out.println(teste--);
```

Operadores Acréscimo e Decréscimo (3)

- Mais exemplos:

```
int a = 5;
int x = a++;// e se fosse: (a++)?

int b = 20;
int y = b+=3 - 2; // e se fosse: (b+=3) - 2?

int c = 10;
int z = c++ + ++c; // e se fosse: (c++) + ++c?
```

Operador Condicional

- Retorna um entre dois valores, baseando-se na resposta da avaliação de uma condição
- Se a expressão for **true**
  - Retorna o valor **depois de "?"**
- Se a expressão for **false**
  - Retorna o valor **depois de ":"**
- É um operador **ternário**
- Exemplo:
  - `int y = (2 > 3) ? 4 : 5`
  - // Qual será o valor de y após a execução desta linha?

Operador Condicional (2)

- Sintaxe:
  - x = (expressão booleana) ? valor a atribuir se **true :** valor a atribuir se **false**
- Exemplos:
  - `int y = (x < 1) ? x*x : 2-x;`
    - Se x < 1, y assume x*x
    - Caso contrário, y assume 2-x

```
int num = 5;
String msg = num > 5
? "É maior que 5"
: "É um número menor ou igual a 5";
```

Operador Condicional (2)

- Aninhamento de operadores condicionais em uma única instrução
  - x = (expressão booleana) ? valor se **true** : (expressão booleana) ? valor se **true** : valor a atribuir se **false**
- Exemplo:

```
int num = 5;
String msg = num > 5
? "É maior que 5"
: num > 0
? "É maior que zero e menor que 5"
: "É um número negativo";
```

Operador Lógico Não/*Not* (!)

Operador	Função
!	Negar um valor booleano

- Inverte o valor de uma variável ou expressão booleana
- Exemplos:
  - `boolean a = true;`
  - `boolean b = !a;`
  - `boolean c = !true;`
  - `boolean d = !(10 > 5);`

Operador Lógico E (&&)

- Para expressão ser considerada verdadeira, ambos os operandos precisam ser **true** (obrigatoriamente)
- Se o primeiro (lado esquerdo da expressão) **não for true**, a operação não terá chances de passar pela validação.
  - Então ele **não verifica** o segundo (lado direito da expressão)
- Exemplo:

```
boolean b = (2 < 3) && (3 < 4);
```

Operador Lógico OU (||)

- Funciona de forma similar ao &&, mas com a diferença que para ser true, **pelo menos um** dos lados da expressão deve ser **true**
- Então, se o primeiro lado da expressão for **true**, ele **não validará o segundo lado**, pois já é suficiente para passar pela avaliação
- Exemplo:

```
int z = 5;
if (++z  >  5  ||  ++z   >  6)
     z++;

//z=7   depois deste código.
```

Exercício Prático

```
int a = 3;
int b = 1;
int c = b + 1;
int d = c > ++b ?  (a - 3) : b;
int e = 4;

if( d > a && b + 1 < a || b > 1)
      c += 7;
if( -7 >= -c)
      a -= 4;
else
      d *= 2;
e = ++d % 3;
b = d++ % 3;
```

Quais são os valores para as variáveis *a*, *b*, *c*, *d* e *e*?