DCC205 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Aula 03

Carlos Bruno Oliveira Lopes
carlosbrunocb@gmail.com

Variáveis em Java

Definição:

- Espaços ou alocações de memória nas quais os dados são armazenados.
  - Em geral, elas devem ter:
    - Nome;
    - Pertencer a um tipo de dado;
    - Receber um valor;
  - As regras para nomes de variáveis em Java são:
    - Podem começar com letra minúscula ou maiúscula, underscore (_), ou símbolo de dólar ($);
    - Não pode conter pontuação nem espaços;
    - Não pode ser utilizados palavras reservadas;

Variáveis em Java

- Tabela de palavras reservadas em JAVA

abstract	boolean	break	byte	case	catch	char
class	const	continue	default	do	double	else
extends	final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long	native
new	package	private	protected	public	return	short
static	strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while	

Variáveis em Java

- Exemplos de nomes válidos para variáveis:

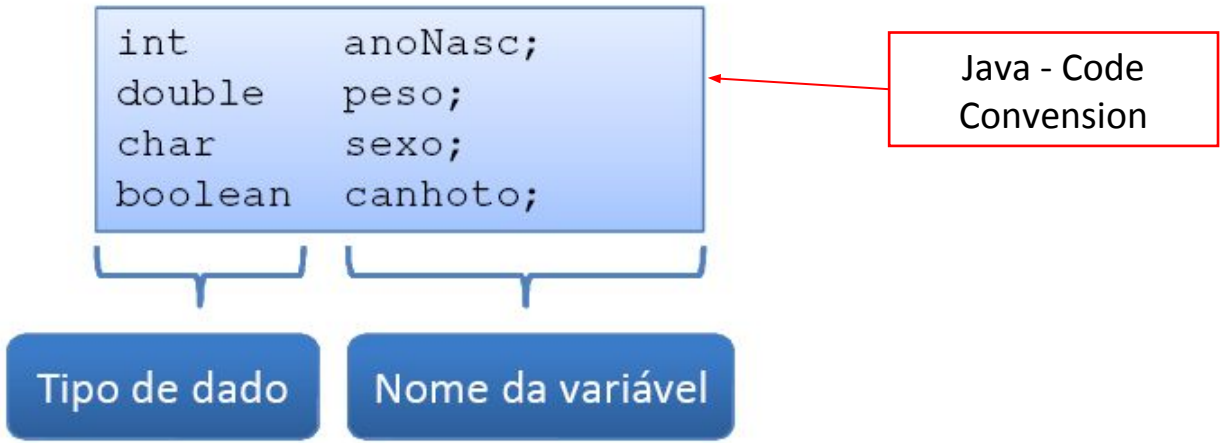
  codigo;
  quantidade;
  indice1;

- Exemplos de nomes não válidos para variáveis:

  1teste;
  nome cliente;
  Código;

Declarando e Usando Variáveis

- Variáveis devem possuir um tipo e um nome



Atribuindo valores às variáveis

- Pode-se atribuir diretamente o valor que variável irá assumir;
- Definir que um variável ira assumir o valor de outra;
- Atribuir a variável o resultado de um expressão, procedimentos ou funções;
  - Ex.:
    - double preco = 12.99;
    - boolean estaAberto = true;
    - double preco = 12.99, valorConta = 5000.10;
    - int numero1 = 10;
    - int numero2 = numero1;
    - float a = 12.0F;
    - float b = 5.2F;
    - float c = (a*b);

Declarando e Usando Variáveis

```
int contador = 20;
int novoContador = contador + 1;
```

novoContador = 21

```
int x = 15;
x = x + 1;
```

x = 16

```
int y = x + x - 10;
```

y = 22

```
int um = 5 % 2; // 5 dividido por 2 dá 2 e tem resto 1;
                // o operador % pega o resto da divisão inteira
```

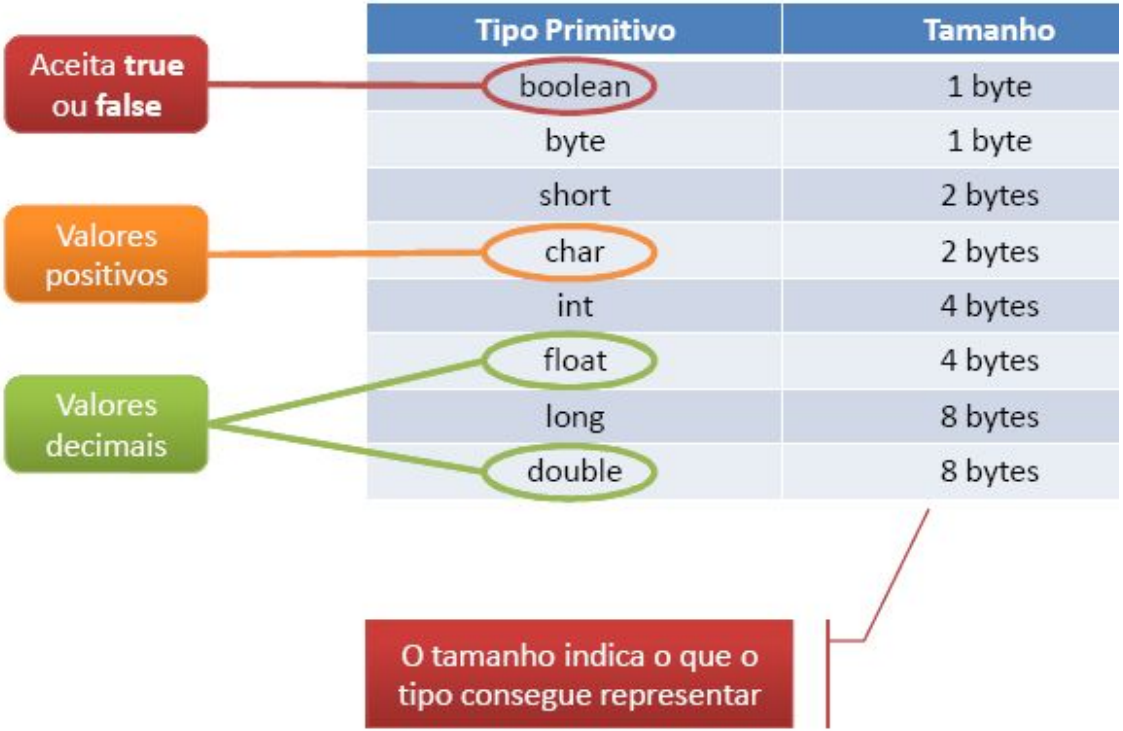
O Java não inicializa as variáveis automaticamente. Caso você não inicialize uma variável e tente usá-la, haverá um erro de compilação.

Tipos de dados

- Tipos primitivos representam valores simples que as variáveis podem assumir;
- Os tipos primitivos são oitos:
  - byte, short, int, long, float, double, char e boolean;

Inteiros	Ponto Flutuante	Caractere	Lógico
byte	float	char	boolean
short	double		
int			
long			

Tipos Primitivos



Tipos inteiros

- Descrição dos tipos inteiros em Java

Tipo	N° de bites	Valor mínimo	Valor máximo
byte	8	-128	128
short	16	-32768	32768
int	32	-2147483648	2147483648
long	64	-9223372036854775808	9223372036854775808

Tipos ponto flutuante

- Descrição dos tipos pontos flutuantes em Java

Tipo	N° de bites	Valor mínimo	Valor máximo
float	32	1.4e-45f	3.4028235e+36f
double	64	4.9e-324	1.7976931348623157e+308

Tipos caractere

- Representa caracteres individualmente, sendo que cada char ocupa 16 bits (2 bytes);
  - Representação do caractere “A” usando caracteres especiais;

Representação	Siginificado
A	Representação explicita
\x41	Representação em hexadecimal
\0101	Representação em octal
\u0041	Representação em UNICODE

Tipos caractere

- Caracteres de uso reservado na linguagem Java

Representação	Siginificado
\n	Nova linha
\r	Retorno de carro
\b	Backspace
\t	Tabulação
\f	Nova página
\’	Apóstrofo
\’’	Aspas
\\	Barra inversa
\unnn	Caractere unicode em hexadecimal nnnn, podendo ir 0000 até FFFF
\0nnn	Caractere octal xxx, podendo ir 000 até 377
\xnn	Caractere hexadecimal representando caracteres de 00 até FFF

Tipo lógico

- Em Java dispomos do tipo boolean que é capaz de receber **false** (falso) ou **true** (verdadeiro);

Tipos Primitivos

Tipo	Quantidade de Bits	Exemplo
char	16	‘a’
byte	8	00000001
int	32	1
short	16	1
long	64	1
float	32	2.99
double	64	2.99
boolean	8	true

Operadores

Aritméticos

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de Divisão (de inteiros)
++	Incremento (inteira e ponto flutuante)
--	Decremento (inteira e ponto flutuante)

Comparação

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

Lógicos

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Outros operadores importantes

Operador	Descrição	Exemplo
++	Incremento	x++
--	Decremento	x--
+=	Soma com valor e atribui o resultado à própria variável	x += 2
-=	Subtrai do valor e atribui o resultado à própria variável	x -= 5
*=	Multiplica pelo o valor e atribui o resultado à própria variável	x *= 3
/=	Divide pelo valor e atribui o resultado à própria variável	x /= 4

Tipos de dados de referência

- Todos os dados manipulados em Java fazem parte de alguma classe;
  - Exceto os dados primitivos da linguagem;

- A instância de classe (ou seja, objeto) criada é sempre manipulada por meio de algo que a referencie – no caso, as variáveis declaradas no programa;

  - Essas variáveis se utilizam da possibilidade de fazer referencias usando o endereço da instância;
    - Esse endereço é atribuído a alguma variável declarada;

      Ex.:

      String str = new String (“Faculdade Java”);

      declaramos uma variável chamada srt, que tem a função de referenciar uma instancia da classe String.

Uso de comandos

- Comando **new**
  - Usado sempre que precisa criar uma nova instância de uma classe (ou seja, um objeto).
- Comando **this**
  - Referência ao próprio objeto que o está utilizando;
    - Ex.:

```
public class Teste {
        private int Numero;
        public void Altera(int Numero) {
                this.Numero = Numero;
        }
}
```

Uso de comandos

- Comando **instanceof**
  - Verifica se um objeto é uma instância da classe especificada.
    - Ex.:

```
Button botão = new Button(“Botão”);
if (botao instanceof Object) {
        System.out.println(“O botão foi gerado a partir da classe Object!”);
}
```

Estruturas de controle

Qualificador	Significado
public	Indica que o conteúdo da classe pode ser utilizado tanto pela classe com por outra que faça referência ao objeto.
protected	Indica que a classe somente pode ser referenciada por métodos que estejam dentro do mesmo pacote, ou em membros da própria classe.
private	Indica acesso restrito a membros da mesma classe.
abstract	Indica que a classe possui métodos abstratos, ou que ela não implementou todos os métodos de uma interface que faça referencia a ela.
final	Esse qualificador faz com que a classe não possa servir de base para herança a outra classe.
strictfp	Indica que todos os valores utilizados nos métodos da classe serão transformados em valores de ponto flutuante, com valores temporários. Ou seja, internamente serão convertidos ou para float ou para Double, em operações com expressões.
static	Permite a utilização de campos estáticos mesmo que a classe seja declarada dentro de outra classe. Permite também que uma classe externa crie um objeto do tipo dessa classe.

Como rodar esses códigos?

```
class TestaIdade {

    public static void main(String[] args) {
        // imprime a idade
        int idade = 20;
        System.out.println(idade);

        // gera uma idade no ano seguinte
        int idadeNoAnoQueVem;
        idadeNoAnoQueVem = idade + 1;

        // imprime a idade
        System.out.println(idadeNoAnoQueVem);
    }
}
```

Constantes em Java

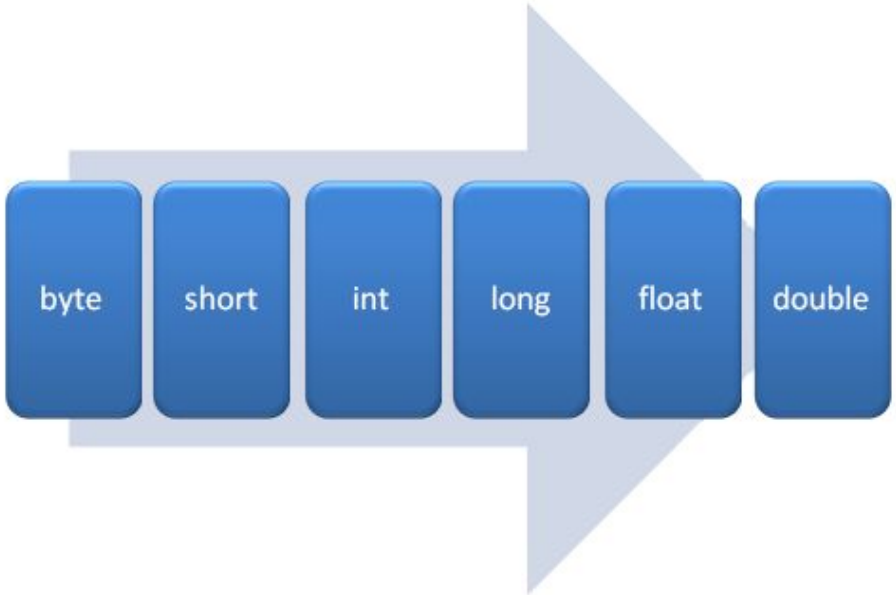
- Constantes são unidades de software que não têm seu valor alterado durante a execução do programa.
- Java não tem constantes, mas tem variáveis **final** que não permitem a alteração de seus valores. A sintaxe é:

final <tipo> <nome_variavel>;

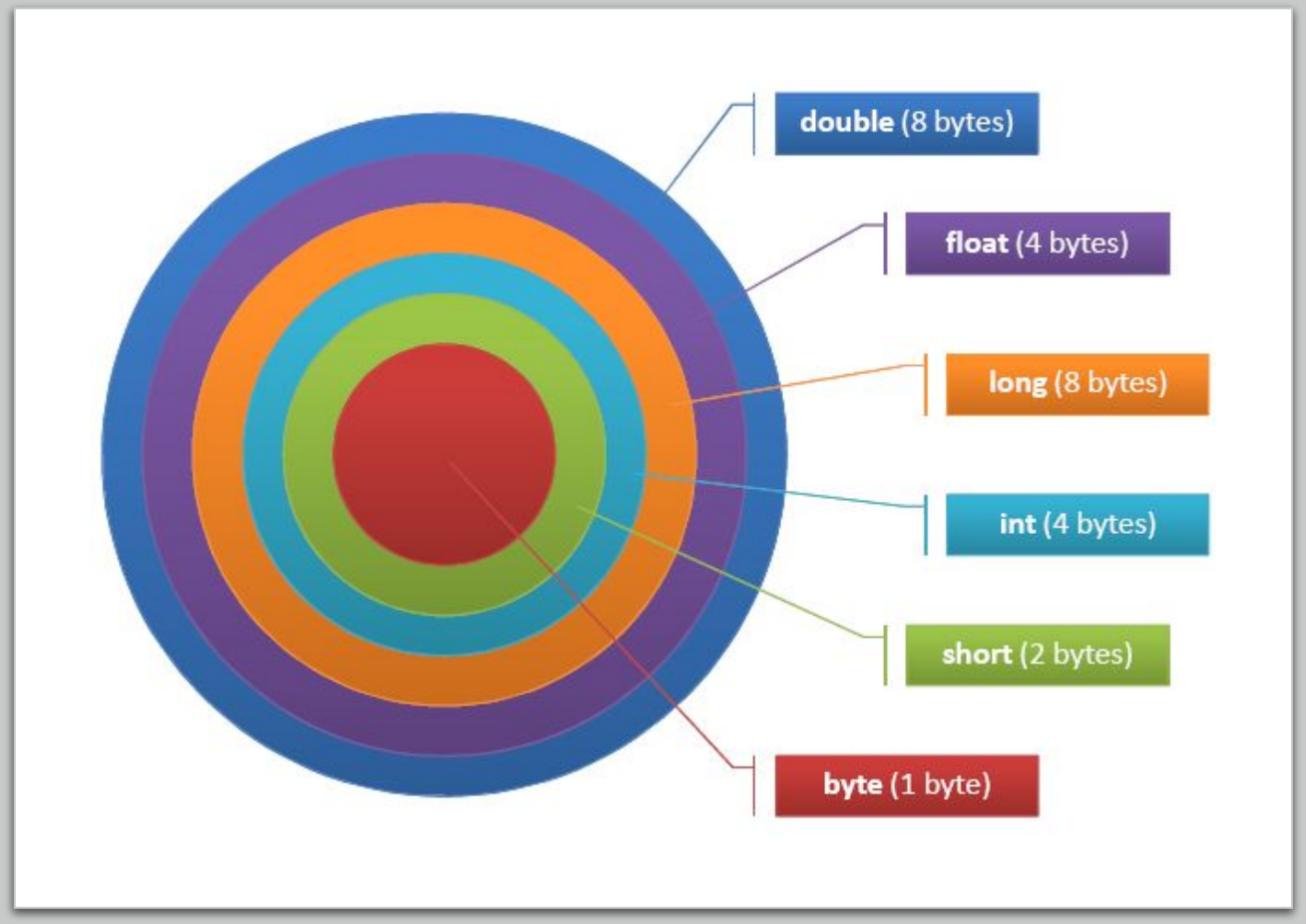
Ex: final double z;

Casting Implícito ou Promoção

- O Java faz a conversão do tipo de dado automaticamente



Casting Implícito ou Promoção



Exemplos de casting implícitos

```
long n1 = 10;
```

10 é do tipo **int** e pode ser atribuído a uma variável **long**

```
float n2 = 5L;
```

5L é do tipo **long** e pode ser atribuído a uma variável **float**

```
double n3 = 2.3f;
```

2.3f é do tipo **float** e pode ser atribuído a uma variável **double**

```
int n4 = 3.5;
```

3.5 é do tipo **double** e não pode ser atribuído a uma variável **int**. É necessário um casting explícito.

Casting e Promoção

- Alguns valores são incompatíveis se você tentar fazer uma atribuição direta.

```
double d = 3.1415;
int i = d; // não compila
```

  - O mesmo ocorre no seguinte trecho:

```
int i = 3.14;
```

  - O mais interessante, é que nem mesmo o seguinte código compila:

```
double d = 5; // ok, o double pode conter um número inteiro
int i = d; // não compila
```

Casting e Promoção

- O código abaixo compila sem problemas

```
int i = 5;
double d2 = i;
```

  - Podemos ordenar que um número real seja moldado(casted) para inteiro:

```
double d3 = 3.14;
int i = (int) d3;
```

Casting Explícito

- A conversão deve ser feita pelo programador

```
double d = 100.0;
int i = d;
```

```
double d = 100.0;
int i = (int) d;
```

- Cuidado com o casting explícito!

```
int n1 = (int) 3.5;
```

O resultado é **3**.
Como o *int* não armazena a parte decimal, ela é perdida.

```
byte n2 = (byte) 129;
```

O resultado é **-127**.
O número 129 é muito grande para caber dentro de uma variável do tipo *byte*.

O Tipo de Dados char

- O char é o único tipo primitivo em Java sem sinal
- Um char indica um caractere, sendo utilizadas aspas simples na sua representação

```
char c = 'A';
```

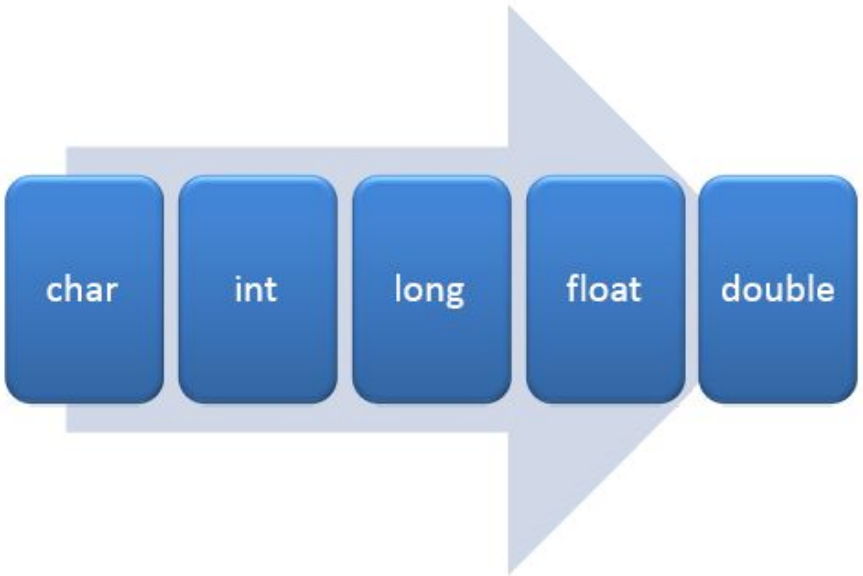
- A atribuição de números a um char também é válida

```
char c = 65;
```

Código ASCII do 'A'

O Tipo de Dados char

- O cast implícito ocorre a partir do tipo int



Castings Possíveis

PARA: / DE:	byte	short	char	int	long	float	double
byte	----	*Impl.*	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
short	(byte)	----	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
char	(byte)	(short)	----	*Impl.*	*Impl.*	*Impl.*	*Impl.*
int	(byte)	(short)	(char)	----	*Impl.*	*Impl.*	*Impl.*
long	(byte)	(short)	(char)	(int)	----	*Impl.*	*Impl.*
float	(byte)	(short)	(char)	(int)	(long)	----	*Impl.*
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

Pós e Pré Incremento ++

- i = i + 1 pode realmente ser substituído por i++. Por exemplo uma atribuição:

```
int i = 5;
int x = i++;
```

Pós incremento

```
int i = 5;
int x = ++i;
```

Pré incremento

Qual é o valor de x? O de i, após essa linha, é 6.

- O operador ++, quando vem após a variável, retorna o valor antigo, e incrementa (pós incremento), fazendo x valer 5.
- Se você tivesse usado o ++ antes da variável (pré incremento), o resultado seria 6:

If

- A sintaxe do if no Java é a seguinte:

Uma condição booleana é qualquer expressão que retorne true ou false. Para isso, você pode usar os operadores <, >, <=, >= e outros. Um exemplo:

```
if (condicaoBooleana) {
    codigo;
}
```

```
int idade = 15;
if (idade < 18) {
    System.out.println("Não pode entrar");
}
```

O If e o Else

- Você pode usar a cláusula **else** para indicar o comportamento que deve ser executado no caso da expressão booleana ser falsa:

```
int idade = 15;
if (idade < 18) {
    System.out.println("Não pode entrar");
} else {
    System.out.println("Pode entrar");
}
```

O If e o Else

Você pode concatenar expressões booleanas através dos operadores lógicos "E" e "OU".O"E" é representado pelo && e o "OU" é representado pelo ||.

```
int idade = 15;
boolean amigoDoDono = true;
if (idade < 18 && amigoDoDono == false) {
    System.out.println("Não pode entrar");
}
else {
    System.out.println("Pode entrar");
}
```

O If e o Else

- O Código anterior ficaria mais legível assim:

```
int idade = 15;
boolean amigoDoDono = true;
if (idade < 18 && !amigoDoDono) {
    System.out.println("Não pode entrar");
}
else {
    System.out.println("Pode entrar");
}
```

O If e o Else

- Para comparar se uma variável tem o **mesmo valor** que outra variável ou valor, utilizamos o operador ==.

```
int mes = 1;
if (mes == 1) {
    System.out.println("Você deveria estar de férias");
}
```

If-else utilizando operador ternário

- Outra possibilidade é utilizar o operador ternário para substituir o if-else

```
int x = 50;
boolean r;

if (x > 30) {
    r = true;
} else {
    r = false;
}
```

```
int x = 50;
boolean r;

r = x > 30 ? true : false;
```

Resultado, se verdadeiro

Resultado, se falso

Estruturas de Controle: switch

- A estrutura switch funciona de forma semelhante a um if-else
- Caso o código entre num bloco case que não possua break, todos os cases abaixo serão executados até que um break seja encontrado;
- Nesta situação, inclusive o bloco default é executado;
- O bloco default é semelhante ao bloco else;

```
int i = 1;

switch (i) {
    case 1:
        System.out.println("Valor = 1");
        break;
    case 2:
        System.out.println("Valor = 2");
        break;
    default:
        System.out.println("Valor não reconhecido");
}
```

Estruturas de Controle: switch

```
public static void main(String[] args) {

    Days day = Days.SATURDAY;
    switch (day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
        System.out.println("boring");
        break;
    case THURSDAY:
    case FRIDAY:
        System.out.println("getting better");
        break;
    case SATURDAY:
    case SUNDAY:
        System.out.println("much better");
        break;
    default: System.out.println("Something like that...");
}

public enum Days {
  MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
  FRIDAY, SATURDAY, SUNDAY
}
```

While

- É um comando usado para fazer um laço (loop), isto é, repetir um trecho de código algumas vezes. A ideia é que esse trecho de código seja repetido enquanto uma determinada condição permanecer verdadeira.

```
int idade = 15;
while (idade < 18) {
    System.out.println(idade);
    idade = idade + 1;
}
```

Do while

- Semelhante ao while
- A condição é testada no fim do bloco

```
int contador = 10;
do {
    System.out.println(contador);
    contador = contador + 1;
} while (contador < 20);
```

- Imprime de 10 a 19.

FOR

- Semelhante ao while, mas possui seção para declaração de variáveis para o loop

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```
for (;;) {
    System.out.println("loop infinito");
}
```

Controlando Loops

- Apesar de termos condições booleanas nos nossos laços, em algum momento, podemos decidir parar o loop por algum motivo especial sem que o resto do laço seja executado.

```
for (int i = x; i < y; i++) {
    if (i % 19 == 0) {
        System.out.println("Achei um número divisível por 19 entre x e y");
        break;
    }
}
```

Controlando Loops

- É possível obrigar o loop a executar o próximo laço. Para isso usamos a palavra chave continue.

```
for (int i = 0; i < 100; i++) {
    if (i > 50 && i < 60) {
        continue;
    }
    System.out.println(i);
}
```

O código acima não vai imprimir alguns números. (Quais exatamente?)

Escopo das variáveis

- No Java, podemos declarar variáveis a qualquer momento. Porém, dependendo de onde você as declarou, ela vai valer de um determinado ponto a outro.

```
// aqui a variável i não existe
int i = 5;
// a partir daqui ela existe
```

  - O escopo da variável é o nome dado ao trecho de código em que aquela variável existe e onde é possível acessá-la.
  - Quando abrimos um novo bloco com as chaves, as variáveis declaradas ali dentro só valem até o fim daquele bloco.

Escopo das variáveis

- Se você tentar acessar uma variável fora de seu escopo, ocorrerá um erro de compilação.

```
// aqui a variável i não existe
int i = 5;
// a partir daqui ela existe
while (condicao) {
    // o i ainda vale aqui
    int j = 7;
    // o j passa a existir
}
// aqui o j não existe mais, mas o i continua dentro do escopo
```

```
EscopoDeVariavel.java:8: cannot find symbol
symbol  : variable j
location: class EscopoDeVariavel
                System.out.println(j);
                                   ^
1 error
```

Escopo das variáveis

- O mesmo vale para um if. Aqui a variável i não existe fora do if e do else!

```
if (algumBooleano) {
    int i = 5;
}
else {
    int i = 10;
}
System.out.println(i); // cuidado!
```

- Então o código para compilar e fazer sentido fica:

```
int i;
if (algumBooleano) {
    i = 5;
}
else {
    i = 10;
}
System.out.println(i);
```

Escopo das variáveis

- Uma situação parecida pode ocorrer com o for:

```
for (int i = 0; i < 10; i++) {
    System.out.println("olá!");
}
System.out.println(i); // cuidado!
```

- Neste for, a variável i morre ao seu término, não podendo ser acessada de fora do for. Precisaria de algo como:

```
int i;
for (i = 0; i < 10; i++) {
    System.out.println("olá!");
}
System.out.println(i);
```

Um Bloco dentro do outro

- Um bloco também pode ser declarado dentro de outro. Isto é, um if dentro de um for, ou um for dentro de um for, algo como:

```
while (condicao) {
    for (int i = 0; i < 10; i++) {
        // código
    }
}
```

Exercício em Sala

1. Faça um aplicativo Java que receba dois números e mostre qual deles é o maior.
2. Ler um numero real, se o número for positivo imprima o dobro dele, senão imprima o número ao quadrado.
3. Receber o salário de um trabalhador e o valor da prestação de um empréstimo, se a prestação for maior que 20% do salário imprima: Empréstimo não concedido, caso contrário imprima: Empréstimo concedido.
4. Elabore um programa que faça leitura de vários números inteiros, até que se digite um número negativo. O programa tem que retornar o maior e o menor número lido.
5. Faça um aplicativo java que determine o mostre os cinco primeiros múltiplos de 3, considerando números maiores que 0.
6. Faça um aplicativo java que leia um número inteiro N e depois imprima os primeiros números naturais ímpares até N.

Exercício de leitura

- Leia o capítulo 3 da apostila fj11 – Orientação a objetos básica:
  - http://www.caelum.com.br/apostila-java-orientacao-objetos/

- Leitura complementar:
- Capítulo 2 – Introdução a classes e objetos do livro:
  - DEITEL, Harvey M. e DEITEL, Paul J. Java - Como Programar, 8ª edição. Pearson. 2010.

Atividade Complementar

1. Vimos apenas os comandos mais usados para controle de fluxo. O Java ainda possui o do...while e o switch. Pesquise sobre eles e diga quando é interessante usar cada um deles.
2. O que acontece se você tentar dividir um número inteiro por 0? E por 0.0? Justifique os resultados
3. Além dos operadores de incremento, existem os de decremento, como --i e i--. Além desses, você pode usar instruções do tipo i += x e i -= x, o que essas instruções fazem? Teste.

Exercício: Fixação de Sintaxe

- Para cada exercício, crie um novo arquivo com extensão .java, e declare aquele cabeçalho, dando nome a uma classe e com um bloco main dentro dele:

```
class ExercicioX {
    public static void main(String[] args) {
        // seu exercício vai aqui
    }
}
```

Não copie e cole de um exercício já existente!
Aproveite para praticar!

Atividade para entregar pelo google sala de aula

1. Imprima todos os números de 150 a 300.
2. Imprima a soma de 1 até 1000.
3. Imprima todos os múltiplos de 3, entre 1 e 100.
4. Imprima os fatoriais de 1 a 10.
5. Faça um for que inicie uma variável n (número) como 1 e fatorial (resultado) como 1 e varia n de 1 até 10:

```
int fatorial = 1;
for (int n = 1; n <= 10; n++) {


}
```

6. No código do exercício anterior, aumente a quantidade de números que terão os fatoriais impressos, até 20, 30, 40. Em um determinado momento, além desse cálculo demorar, vai começar a mostrar respostas completamente erradas. Por quê? Mude de int para long para ver alguma mudança.

Atividade para entregar pelo google sala de aula

7. Imprima os primeiros números da série de Fibonacci até passar de 100. A série de Fibonacci é a seguinte: 0, 1, 1, 2, 3, 5, 8, 13, 21, etc... Para calculá-la, o primeiro elemento vale 0, o segundo vale 1, daí por diante, o n-ésimo elemento vale o (n-1)-ésimo elemento somado ao (n-2)-ésimo elemento (ex: 8 = 5+3).
8. Escreva um programa que, dada uma variável x com algum valor inteiro, temos um novo x de acordo com a seguinte regra:
   - se x é par, x = x / 2
   - se x é impar, x = 3 * x + 1
   - imprime x
   - O programa deve parar quando x tiver o valor final de 1. Por exemplo, para x = 13, a saída será:40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

Atividade para entregar pelo google sala de aula

Obs: Imprimindo sem pular linha:

- Um detalhe importante é que uma quebra de linha é impressa toda vez que chamamos println.
- Para não pular uma linha, usamos o código a seguir:
  System.out.print(variavel);

10. Imprima a seguinte tabela, usando fors encadeados:

```
1
2 4
3 6 9
4 8 12 16
n n*2 n*3 .... n*n
```