Programação Orientada a Objetos

Professor Filipe Dwan Pereira

Aula 2 – Variáveis Primitivas, casting e controle de fluxo de dados

"Melhor o longânimo do que o valente, e o que governa o seu espírito do que o que toma uma cidade."

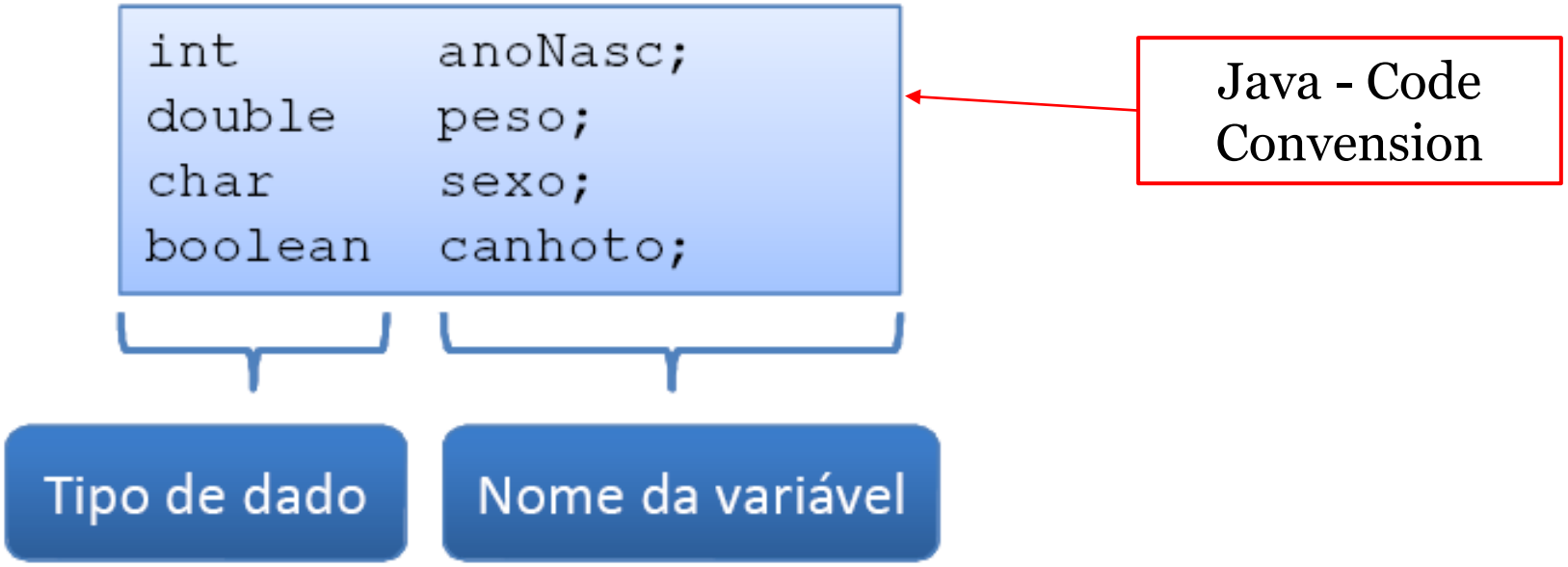
Provérbios 16:32

Disclaimer

- Este slide foi baseado nas seguintes fontes principais:
  - SOFTBLUE. Professor Carlos Eduardo Gusso Tosin. Fundamentos de Java. http://www.softblue.com.br/.
  - Slides professor Horácio Oliveira – UFAM.
  - CAELUM. Java e Orientação a Objetos. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos/
  - K19. Java e Orientação a Objetos. Disponível em: http://www.k19.com.br/cursos/orientacao-a-objetos-em-java.

Declarando e Usando Variáveis

- Variáveis devem possuir um tipo e um nome



Comentários em Java

Para fazer um comentário em java, você pode usar o // para comentar até o final da linha, ou então usar o /* */ para comentar o que estiver entre eles.

Para comentar várias linhas e gerar documentação com Javadoc: /** */

```
/* comentário daqui,
ate aqui */

// uma linha de comentário sobre a idade
int idade;
```

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.

Como rodar esses códigos?

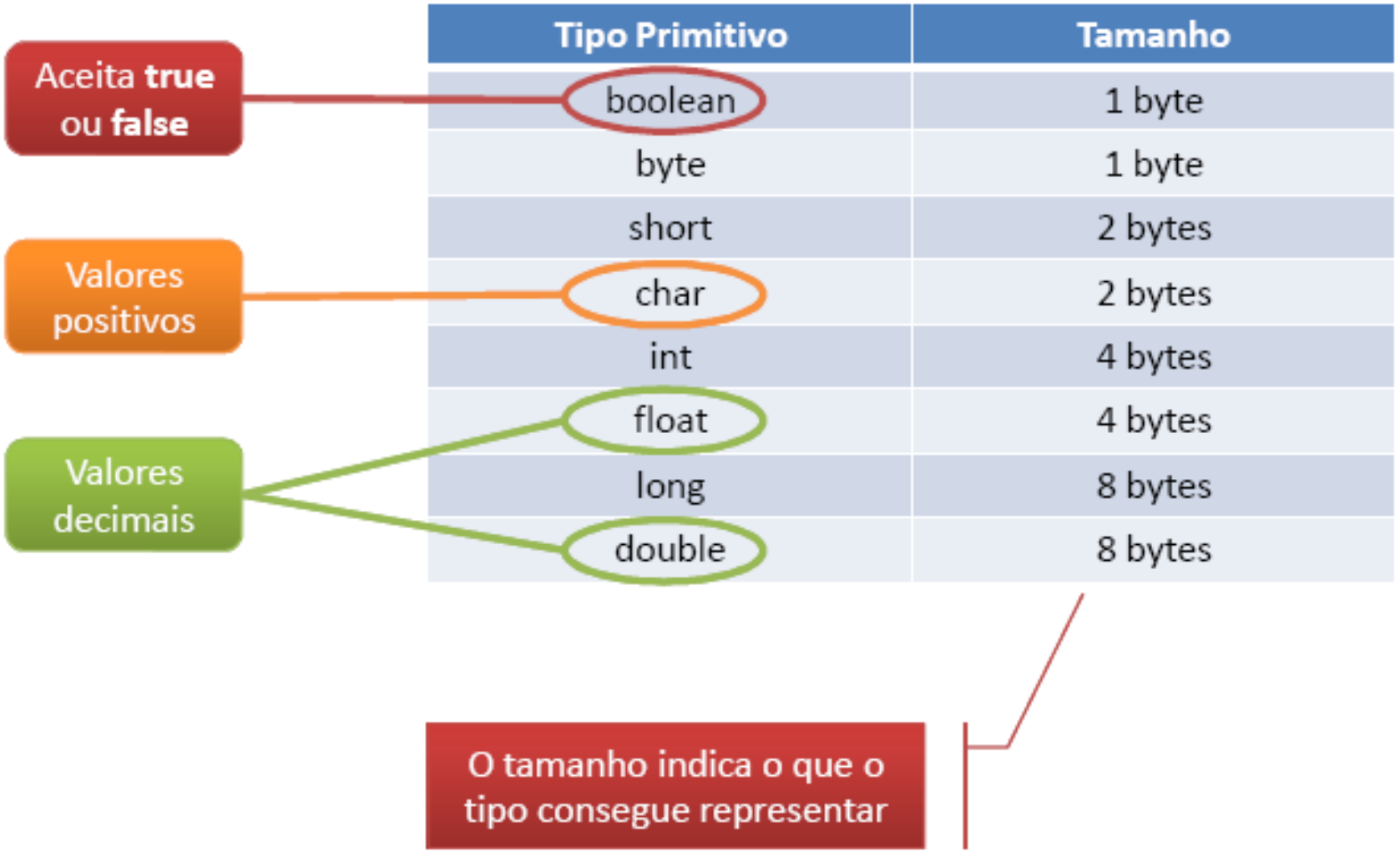
```
class TestaIdade {

    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);
    }
}
```

Tipos Primitivos



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)

Lógicos

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

Comparação

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

Outros operadores importantes

+=	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

Declarando e Usando Variáveis

- double armazena um número com ponto flutuante (e que também pode armazenar um número inteiro):

```
double pi = 3.14;
double x = 5 * 10;
```

- O tipo boolean armazena um valor verdadeiro ou falso:

```
boolean verdade = true;

int idade = 30;
boolean menorDeIdade = idade < 18;
```

O tipo de dado 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'

- Variáveis do tipo char são pouco usadas no dia a dia.
- Veremos mais a frente o uso das Strings, que usamos constantemente, porém estas não são definidas por um tipo primitivo.

Tipos Primitivos e Valores

- Esses tipos de variáveis são tipos primitivos do Java: **o valor que elas guardam são o real conteúdo da variável**. Quando você utilizar o operador de atribuição = o valor será copiado.

```
int i = 5; // i recebe uma cópia do valor 5
int j = i; // j recebe uma cópia do valor de i
i = i + 1; // i vira 6, j continua 5
```

Constantes em Java

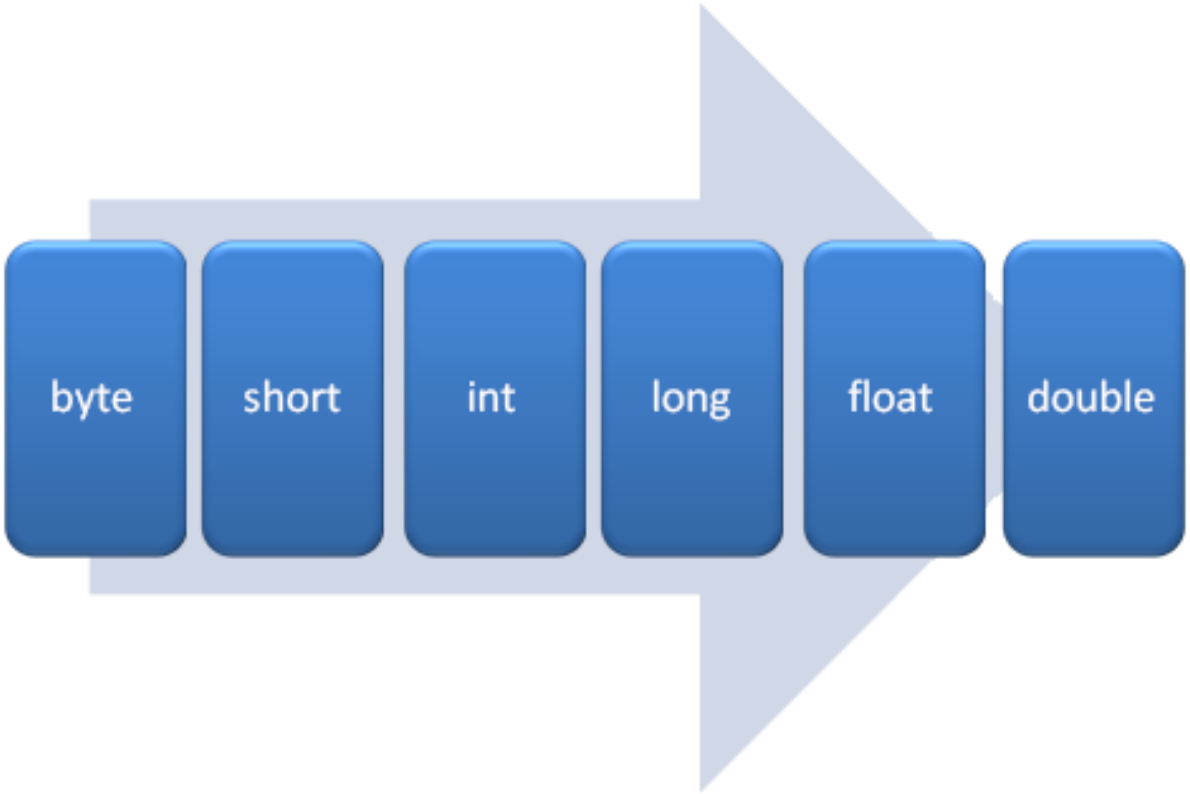
- Constantes são unidades de software que não têm seu valor alterado durante a execução do programa.
- Java não tem constantes, mas tem variáveis **final** que não permitem a alteração de seus valores. A sintaxe é:

final <tipo> <nome_variavel>;

Ex: final double z;

Casting Implícito ou Promoção

- O Java faz a conversão do tipo de dado automaticamente



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

- Nem mesmo o seguinte código compila:

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

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*.

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)	----

Obs: 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-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

```
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");
}
```

- 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

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
}
```

O While

- O 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.

O 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

```
// 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
```

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

```
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);
```

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 para entregar pelo SIGAA

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 SIGAA

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 SIGAA

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);

9. Imprima a seguinte tabela, usando fors encadeados:

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

Referências Bibliográficas

- http://docs.oracle.com/javase/7/docs/technotes/guides/language/strings-switch.html
- DEITEL, Harvey M. e DEITEL, Paul J. Java - Como Programar, 8ª edição. Pearson. 2010.
- BLOCH, Joshua. Effective Java, 2ª edição. Addison-Wesley, 2008.
- CAELUM. Java e Orientação a Objetos. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos/. Acesso em: Agosto de 2014.
- K19. Java e Orientação a Objetos. Disponível em: http://www.k19.com.br/cursos/orientacao-a-objetos-em-java. Acesso em: Agosto de 2014.
- HORSTMANN, CORNELL. Core Java Volume I – Fundamentos, 8º Edição. São Paulo, Pearson Education, 2010.
- Aulas professor Carlos Tosin. Softblue. http://www.softblue.com.br/course/home/id/1

“Seja a Mudança que você quer ver no mundo”. Ghandi

filipedwan@gmail.com

;-)

skype filipedwan

twitter @filipedwan