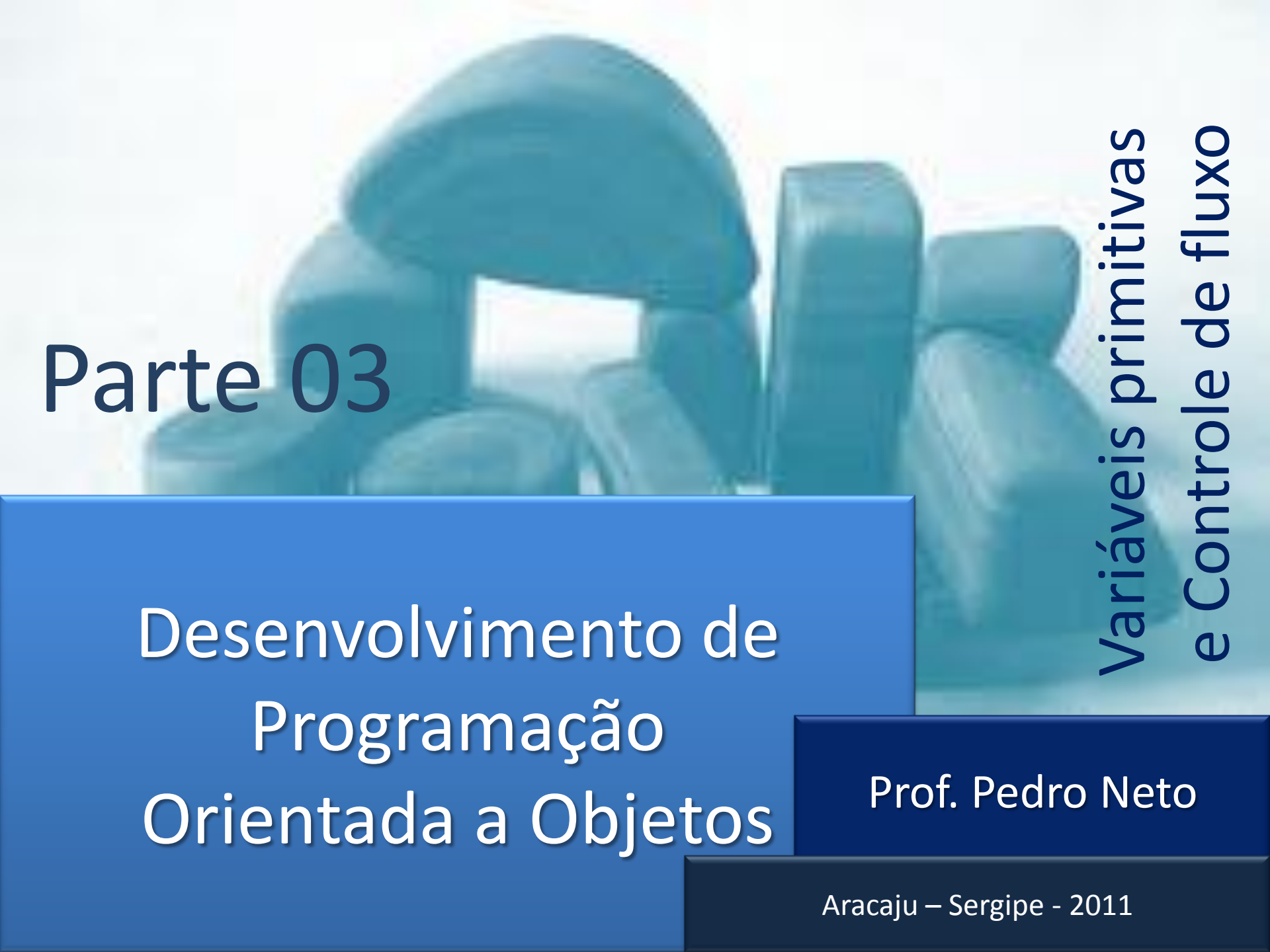

Parte 03

Variáveis primitivas e Controle de fluxo

Desenvolvimento de Programação Orientada a Objetos

Prof. Pedro Neto

Aracaju – Sergipe - 2011

Conteúdo

3 Variáveis primitivas e controle de fluxo

i. Declarando e usando variáveis
ii. Tipos primitivos e valores
iii. Exercícios
iv. Casting e promoção
v. O If-Else
vi. O While
vii. O For
viii. Controlando loops
ix. Escopo das variáveis
x. Um bloco dentro do outro
xi. Um pouco mais...
xii. Exercícios
xiii. Desafios

Declarando e usando variáveis

Dentro de um bloco, podemos declarar variáveis e usá-las. Em Java, toda variável tem um tipo que não pode ser mudado uma vez que declarado:

```
tipoDaVaríavel  nomeDaVariável;
```

Por exemplo, e possível ter uma idade que vale um número inteiro:

```
int idade;
```

Com isso, você declara a variável idade, que passa a existir a partir deste momento. Ela é do tipo `int`, que guarda um numero inteiro. A partir de agora você pode usá-la, primeiro atribuindo valores. A linha a seguir é a tradução de **"idade deve valer agora quinze"**.

```
idade = 15;
```

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.

```
/* comentário daqui,
ate aqui */
//uma linha de comentário sobre a idade
int idade;
```

Alem de atribuir, você pode utilizar esse valor. O código a seguir declara novamente a variável idade com valor 15 e imprime seu valor na saída padrão através da chamada a `System.out.println`.

```
//declara a idade
int idade;
idade = 15;

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

Atribuindo valores a variáveis

Por fim, podemos utilizar o valor de uma variável para algum outro propósito, como alterar ou definir uma segunda variável. O código a seguir cria uma variável chamada **`idadeNoAnoQueVem`** com valor de **`idade`** mais um.

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

No momento que você declara uma variável, também e possível inicializá-la por praticidade:

```
int idade = 15;
```

Operadores Aritméticos

Você pode usar os operadores +,- , / e * para operar com números, sendo eles responsáveis pela adição, subtração, divisão e multiplicação, respectivamente. Além desses operadores básicos, há o operador % (módulo) que nada mais é que o resto de uma divisão inteira. Veja alguns exemplos:

```
int quatro = 2 + 2;
int tres = 5 - 2;
int oito = 4 * 2;
int dezesseis = 64 / 4;
int um = 5 % 2; // 5 dividido por 2 dá 2 e tem resto 1;
// o operador % pega o resto da divisão inteira
```

Testando o Código

Você deve colocar esses trechos de código dentro do método `main`, que vimos no capitulo anterior. Isto é, isso deve ficar no miolo do programa. Use bastante `System.out.println`, dessa forma você pode ver algum resultado, caso contrário, ao executar a aplicação, nada aparecerá.

Por exemplo, para imprimir a idade e a `idadeNoAnoQueVem` podemos escrever o seguinte programa de exemplo:

```
1.class TestaIdade {
2.
3.    public static void main(String[] args) {
4.
5.    // declara a idade
6.    int idade;
7.    idade = 15;
8.
9.    // imprime a idade
10.   System.out.println(idade);
11.
12.   // gera uma idade no ano seguinte
13.   int idadeNoAnoQueVem;
14.   idadeNoAnoQueVem = idade + 1;
15.
16.   // imprime a idade
17.   System.out.println(idadeNoAnoQueVem);
18.
19.   }
20.}
```

Outros tipos de dados

Representar números inteiros é fácil, mas como guardar valores reais, como frações de números inteiros e outros? Outro tipo de variável muito utilizado é o **`double`**, que armazena um número com ponto flutuante.

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

O tipo **`boolean`** armazena um valor verdadeiro ou falso, e só.

```
boolean verdade = true;
```

O tipo **`char`** guarda um e apenas um caractere. Esse caractere deve estar entre aspas simples. Não se esqueça dessas duas características de uma variável do tipo **`char`**! Por exemplo, ela não pode guardar um código como ‘‘ pois o vazio não é um caractere!

```
char letra = 'a';
System.out.println(letra);
```

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
```

Aqui, i fica com o valor de 6. Mas e j? Na segunda linha, j está valendo 5. Quando i passa a valer 6, será que j também fica valendo? Não, pois o valor de um tipo primitivo sempre e copiado. Apesar da linha 2 fazer j = i, a partir desse momento essas variáveis não tem relação nenhuma: o que acontecer com uma não refletirá em nada com a outra.

Vimos aqui os tipos primitivos que mais aparecem. O Java tem outros, que são o byte, short, long e float. Cada tipo possui características especiais que, para um programador avançado, podem fazer muita diferença.

Tipos primitivos no Java

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 represntável 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 4.9406564584124654e-324 e o maior é 1.7976931348623157e+308

Exercícios

1) Na empresa onde trabalhamos, há tabelas com o quanto foi gasto em cada mês. Para fechar o balanço do primeiro trimestre, precisamos somar o gasto total. Sabendo que, em Janeiro foi gasto 15000 reais, em Fevereiro, 23000 reais e em Marco, 17000 reais, faca um programa que calcule e imprima o gasto total no trimestre. Siga esses passos:

a)Crie uma classe chamada Balanço Trimestral com um bloco **`main`**, como nos exemplos anteriores;
b)Dentro do **`main`** (o miolo do programa), declare uma variável inteira chamada **`gastosJaneiro`** e inicialize-a com 15000;
c)Crie também as variáveis **`gastosFevereiro`** e **`gastosMarco`**, inicializando-as Com 23000 e 17000, respectivamente, utilize uma linha para cada declaração;
d)Crie uma variável chamada gastosTrimestre e inicialize-a com a soma das outras 3 variáveis:

```
int gastosTrimestre = gastosJaneiro + gastosFevereiro +
gastosMarco
```

e)Imprima a variável **`gastosTrimestre`**.

Casting e promoções

Alguns valores são incompatíveis se você tentar fazer uma atribuição direta. Enquanto um numero real costuma ser representado em uma variável do tipo **`double`**, tentar atribuir ele a uma variável **`int`** não funciona pois e um código que diz: “i deve valer d”, mas não se sabe se d realmente é um número inteiro ou não.

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

O mesmo ocorre no seguinte trecho:

```
int i = 3.14;
```

O mais interessante, e 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 promoções

Apesar de 5 ser um bom valor para um `int`, o compilador não tem como saber que valor estará dentro desse `double` no momento da execução. Esse valor pode ter sido digitado pelo usuário, e ninguém vai garantir que essa conversão ocorra sem perda de valores. Já no caso a seguir é o contrário:

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

O código acima compila sem problemas, já que um `double` pode guardar um número com ou sem ponto flutuante. Todos os inteiros representados por uma variável do tipo `int` podem ser guardados em uma variável `double`, então não existem problemas no código acima. As vezes, precisamos que um número quebrado seja arredondado e armazenado num número inteiro.

Casting e promoções

Para fazer isso sem que haja o erro de compilação, é preciso ordenar que o número quebrado seja moldado (casted) como um número inteiro. Esse processo recebe o nome de casting.

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

O casting foi feito para moldar a variável d3 como um **`int`**. O valor dela agora é 3. O mesmo ocorre entre valores **`int`** e **`long`**.

```
long x = 10000;
int i = x; // não compila, pois pode estar perdendo informação
```

E, se quisermos realmente fazer isso, fazemos o casting:

```
long x = 10000;
int i = (int) x;
```

O If - Else

A sintaxe do **`if`** no Java é a seguinte

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

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

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

O If - Else

Além disso, você pode usar a clausula **`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 – Else –Operadores Lógicos

Você pode concatenar expressões booleanas através dos operadores lógicos "E" e "OU". O "E" e representado pelo & e o "OU" e 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 – Else –Operadores Lógicos - Negação

Esse código poderia ainda ficar mais legível, utilizando-se o operador de negação, o !. Esse operador transforma uma expressão booleana de false para true e vice versa.

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

O If – Else –Operadores Lógicos – Curto Circuito - && e II

```
if (true | algumaCoisa) {
        // ...
}
```

```
if (true || algumaCoisa) {
        // ...
}
```

O While

O **while** é um comando usado para fazer um laço (loop), isto é, repetir um trecho de código algumas vezes. A idéia e 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;
}
```

O trecho dentro do bloco do **while** será executado até o momento em que a condição idade < 18 passe a ser falsa. E isso ocorrera exatamente no momento em que idade ==18, o que não o fará imprimir 18.

```
int i = 0;
while(i < 10) {
      System.out.println(i);
      i = i + 1;
}
```

Já o **while** acima imprime de 0 a 9.

O for

Outro comando de loop extremamente utilizado é o **`for`**. A idéia é a mesma do **`while`**, fazer um trecho de código ser repetido enquanto uma condição continuar verdadeira. Mas além disso, o **`for`** isola também um espaço para inicialização de variáveis e o modificador dessas variáveis. Isso faz com que fiquem mais legíveis as variáveis que são relacionadas ao loop:

```
for (int i = 0; i < 10; i = i + 1) {
        System.out.println("olá!");
}
```

Pós e Pré incremento

i = i + 1 pode realmente ser substituído por i++ quando isolado, porém, em alguns casos, temos o seguinte:

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

Qual e o valor de x? O de i, após essa linha, é 6. O operador ++, quando vem à frente da 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, como segue:

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

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;
   }
}
```

O código acima vai percorrer os números de x a y e parar quando encontrar um número divisível por 19, uma vez que foi utilizada a palavra chave **`break`**. Da mesma maneira, e 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);
}
```

Escopo de 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 que é possível acessá-la. Quando abrimos um novo bloco com as chaves, as variáveis declaradas ali dentro só valem até o fim daquele bloco.

```
//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 a valer
```

Um bloco dentro do outro

Um bloco também pode ser declarado dentro de outro. Isto e, 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
      }
}
```

Um pouco mais

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 e interessante usar cada um deles.
2) Algumas vezes temos vários laços encadeados. Podemos utilizar o `break` para quebrar o laço mais interno, mas se quisermos quebrar um laço mais externo, teremos de encadear diversos `if` e seu código ficara uma bagunça. O Java possui um artifício chamado labeled loops, pesquise sobre eles.
3) O que acontece se você tentar dividir um numero inteiro por 0? E por 0.0?
4) Existe um caminho entre os tipos primitivos que indicam se hé a necessidade ou não de casting entre os tipos. Por exemplo, int >long >double (um int pode ser tratado como um double, mas não o contrario). Pesquise (ou teste), e posicione os outros tipos primitivos nesse fluxo.
5) Existem outros operadores, como o %, <<, >>. Descubra para que servem.
6) Alem dos operadores de incremento, existem os de decremento, como --i e i--, além desses, você pode usar instrucoes do tipo i += x e i -=x, o que essas instruções fazem?

Exercícios

1)Imprima todos os números de 150 a 300.
2)Imprima a soma de 1 ate 1000.
3)Imprima todos os múltiplos de 3, entre 1 e 100.
4)Imprima os fatoriais de 1 a 10.
5)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. Porque? Mude de int para long, e você poderá ver alguma mudança.
6) Imprima os primeiros números da serie de Fibonacci até passar de 100. A serie de Fibonacci é a seguinte: 0, 1, 1, 2, 3, 5, 8, 13, 21, etc... Para calculá-la, o primeiro e o segundo elementos valem 1, dai por diante, o enésimo elemento vale o n1esimo elemento somando ao n2esimo elemento (ex: 8 = 5 + 3).
7) Escreva um programa que, dada uma variável x (com valor 180, por exemplo), temos y de acordo com a seguinte regra:

se x é par, y = x / 2
se x é impar, y = 3 * x + 1
imprime y
O programa deve então jogar o valor de y em x e continuar até que y tenha o valor final de 1.
Por exemplo, para x = 13, a saída será:
40 >20 >10 >5 >16 >8 >4 >2 >1

Desafio

Façaa o exercício da serie de Fibonacci usando apenas duas variáveis.

Obs.

Um detalhe importante do método que estamos usando até agora e que uma quebra de linha é impressa toda vez que chamado. Para não pular uma linha usamos o método a seguir:

```
System.out.print(variavel);
```

Dados de Contato


79 9949 4098


http://pedroneto.yolasite.com/


pedro@pyxistec.com.br


psneto@emsergipe.com


pedro.pyxistec@gmail.com


http://www.facebook.com/pedro.neto.se


pedropyxis


http://lattes.cnpq.br/4891420246888248