



UNIVERSIDADE  
FEDERAL DO CEARÁ

TÓPICO  
**06**



# PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Marcos Vinicius de Andrade Lima  
E-mail: [marcos.vinicius@ufc.br](mailto:marcos.vinicius@ufc.br)



## Olá!

**Sou Marcos Vinicius**

No tópico passado nós tivemos uma pequena introdução sobre UML...

Neste tópico melhoraremos os métodos dos nossos objetos com **estruturas de controle de fluxo!**

“

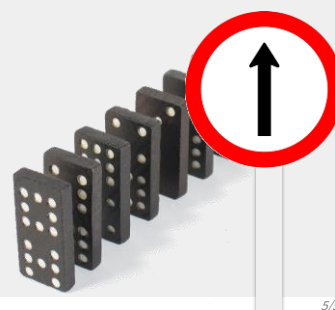
**Seus clientes mais insatisfeitos são  
sua melhor fonte de aprendizado**  
(Bill Gates)



**Condicionais**

## FLUXO DE EXECUÇÃO

- Tudo o que **fizemos até agora** em Java foram **seqüências de comandos**, onde o **fluxo de execução seguia sempre de forma contínua**, ou seja, do início ao fim, executando todos os comandos na ordem que foram inseridos.
- **Será que trabalhando desta forma conseguimos elaborar qualquer programa?**



Prof. Marcos Vinicius – UFC/Russas - POO

5/50

**Não é preciso muito tempo para perceber que uma sequência única no fluxo de execução, sem a utilização de outros caminhos, tornaria impossível a construção de programas mais completos e complexos.**



## ESTRUTURAS DE CONTROLE DE FLUXO



Prof. Marcos Vinicius – UFC/Russas - POO

7/50

## CONDICIONAL

- A estrutura de controle condicional é a estrutura mais simples existente. Ela é utilizada quando é preciso decidir qual caminho o fluxo de execução deve seguir, de acordo com a análise de uma condição.
- **A estrutura condicional sempre possui:**
  - um teste, com uma expressão lógica;
  - uma ação que é realizada quando o resultado do teste é verdadeiro.
- **E também pode ter:**
  - uma ação alternativa, que é realizada quando o resultado do teste é falso.

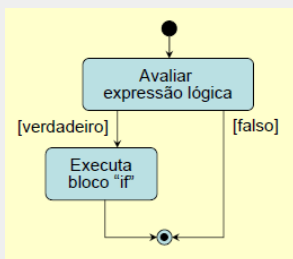


Prof. Marcos Vinicius – UFC/Russas - POO

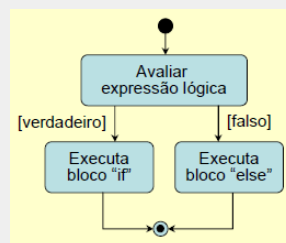
8/50

## CONDICIONAL: IF E IF-ELSE

```
if ( <condição> ) {
    <ações>
}
```



```
if ( <condição> ) {
    <ações>
} else {
    <outras ações>
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

9/50

## OPERADOR TERNÁRIO

- É um recurso para tomada de decisões com objetivo **similar ao do if-else**, entretanto é **codificado em apenas uma linha**

**(condição) ? ação 1 : ação 2;**

- Ao avaliar a condição, caso ela seja verdadeira, a ação 1, declarada após o ponto de interrogação (?) será executada; caso contrário, o programa irá executar a ação 2, declarada após os dois pontos (:)
- IMPORTANTE:** esse operador é utilizado para uma estrutura de decisão simples (iniciar uma variável, retornar um valor ou integrar um bloco de código), ajudando na legibilidade

Prof. Marcos Vinicius – UFC/Russas - POO

10/50

## OPERADOR TERNÁRIO: EXEMPLO PRÁTICO

```
// numeroDias é um valor de 1 a 30
String msg = (numeroDias <= 15) ? "1ª Quinzena" : "2ª Quinzena";
System.out.println(msg);
```

Viu como o código ficou mais legível?



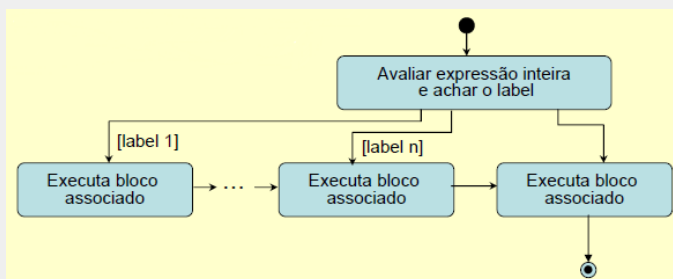
```
String msg = "";
if (numeroDias <= 15){
    msg = "1ª Quinzena";
}
else {
    msg = "2ª Quinzena";
}
System.out.println(msg);
```

Prof. Marcos Vinicius – UFC/Russas - POO

11/50

## SELEÇÃO COM SWITCH

```
switch ( <expressão integral> ) {
    case <label1>: <ações1>
    case <label2>: <ações2>
    ...
    default: <ações default>
}
```

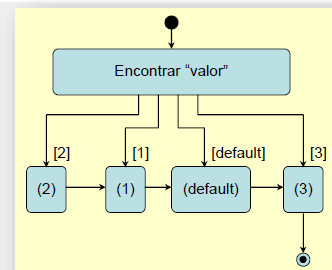


Prof. Marcos Vinicius – UFC/Russas - POO

12/50

## VEJA O SWITCH EM AÇÃO!

```
...
switch ( valor ) {
    case 2:
        System.out.print("(2)");
    case 1:
        System.out.print("(1)");
    default:
        System.out.print("(default)");
    case 3:
        System.out.print("(3)");
}
...
```

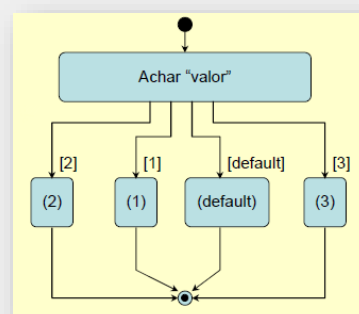


Prof. Marcos Vinicius – UFC/Russas - POO

13/50

## QUE LEGAL, SWITCH + BREAK!

```
...
switch ( valor ) {
    case 2:
        System.out.print("(2)");
        break;
    case 1:
        System.out.print("(1)");
        break;
    default:
        System.out.print("(default)");
        break;
    case 3:
        System.out.print("(3)");
}
...
```



Prof. Marcos Vinicius – UFC/Russas - POO

14/50

## MÉTODOS DE SETTERS

- Agora que você aprendeu sobre os condicionais, poderá controlar os valores atribuídos às variáveis de instância!
- Lembre-se que a classe deve ser a responsável por verificar se o valor recebido pode ou não ser atribuído a uma dada variável de instância
- Trate os métodos de setters com extremo cuidado para evitar problemas em seu código
- **O uso correto de setters evita que o objeto fique em um estado inconsistente!**

Prof. Marcos Vinicius – UFC/Russas - POO

15/50

## SE LIGA NAS IDEIAS...

- O **estado interno dos objetos** deve ser mantido **sempre consistente**  
  
Por exemplo, não gostaríamos de ver em um objeto o valor de uma data com o dia 30 de fevereiro (estado inconsistente)
- As **variáveis de instância** são as responsáveis pelo **estado interno dos objetos**
- O **acesso direto** a estas variáveis pode comprometer o estado interno dos objetos (situações inconsistentes)

Prof. Marcos Vinicius – UFC/Russas - POO

16/50

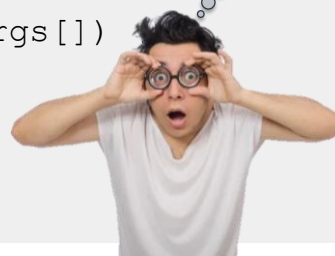


## UM EXEMPLO A SER EVITADO...

```
public class Data{
    int dia;
    int mes;
    int ano;
    ...

    public class Cliente {
        public static void main(String args[])
            Data hoje = new Data();
            hoje.mes = 15;
            ...
    }
}
```

Eu acho que  
tem um  
pequeno *bug*...



Prof. Marcos Vinicius – UFC/Russas - POO

17/50

## OUTRO EXEMPLO PARA PENSAR...

**KABOOM!!!**

- Imagine que você desenvolveu um objeto que controla a pressão de uma caldeira
- O que poderia acontecer se o código abaixo fosse implementado?

```
public void setPressao(float pressão){
    this.pressao = pressão;
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

18/50

**Não é recomendável** deixar que os **outros** sejam **responsáveis** pelas **validações** das **variáveis de instâncias**!

Quem tem que saber se uma data, por exemplo, está correta é a classe especializada em datas!

E saber se uma pressão é permitida é responsabilidade da calceira!



Para evitar que situações como essas ocorram, uma **boa prática é impedir acesso direto às variáveis de instância dos objetos**!



Use o modificador **private** em todas as variáveis de instância!

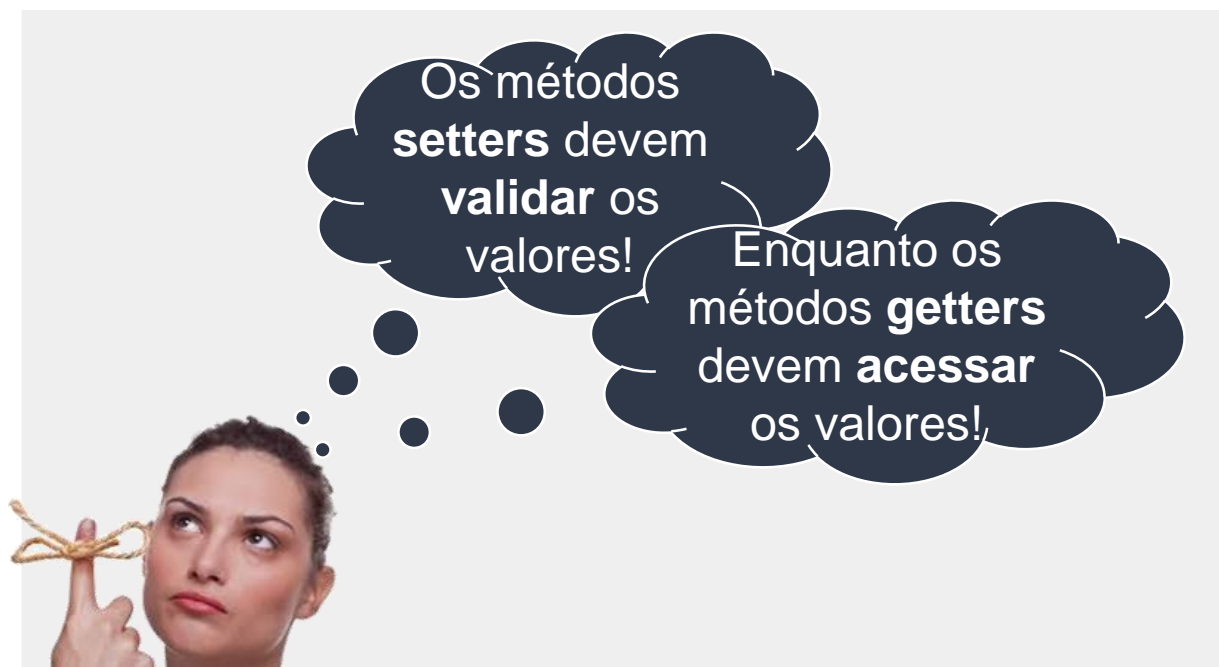
## MODIFICANDO A CLASSE DATA

```
public class Data{  
    private int dia;  
    private int mes;  
    private int ano;  
    ...  
  
public class Cliente{  
    public static void main(String args[]){  
        Data hoje = new Data();  
        hoje.mes = 15;  
        ...  
    }  
}
```

**ERRO DE COMPILAÇÃO:** a classe Cliente agora não pode mais ser compilada!

Prof. Marcos Vinicius – UFC/Russas - POO

21/50



## MÉTODOS DE GETTERS E SETTERS

- Em Java existem algumas **convenções** que padronizam os nomes dos métodos de **acesso às variáveis de instância encapsuladas**
- Eles são chamados de métodos **getters** e **setters**
- **Métodos de alteração do valor → set**  
public void **setNomeDaVariavel**(...)
- **Métodos de obtenção do valor → get**  
public retorno **getNomeDaVariavel**()

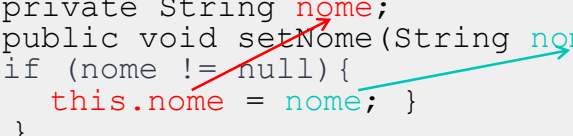
Prof. Marcos Vinicius – UFC/Russas - POO

23/50

## REFERÊNCIA THIS. GERALMENTE É USADA PARA...

- diferenciar nomes de parâmetros e variáveis de instância:  

```
private String nome;
public void setNome(String nome) {
    if (nome != null) {
        this.nome = nome; }
    }
```


- pode ser usada para ativar métodos como uma referência comum:  

```
void atribuirNota( int numProva) {
    this.atribuirNota(numProva, 0.0); }
```
- pode ser passada como parâmetro ou atribuída a outra variável do mesmo tipo:  

```
processar(this);
Estudante e = this;
```

Prof. Marcos Vinicius – UFC/Russas - POO

24/50

## REFERÊNCIA THIS.

- O identificador **this** denota o objeto no qual o método é chamado.
- Cada objeto é referenciado dentro da própria classe através da referência **this**.
- A referência **this** é passada como um **parâmetro implícito** quando um método de instância é chamado, representando sempre o **objeto corrente**.
- **Nunca esqueça: só pode ser usada na classe de definição.**



Prof. Marcos Vinicius – UFC/Russas - POO

25/50

## SOBRECARGA DE MÉTODOS: EXEMPLO

```
public class Data {
    int dia, mes, ano;

    void exibir() { ... }

    int diasDoAno() { ... }

    void incrementaAno() {
        ano = ano + 1; }

    void incrementaAno( int anos ) {
        ano = ano + anos;
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

26/50

## SOBRECARGA DE MÉTODOS

- Assinatura dos métodos:  
**nome + tipos e ordem dos parâmetros.**
- Regra básica para sobrecarga:
  - ✓ Métodos sobrecarregados **não podem ter a mesma assinatura.**
  - ✓ Os parâmetros têm que diferir em tipo, ordem ou número!

Prof. Marcos Vinicius – UFC/Russas - POO

27/50

## E AÍ, ESTÁ TUDO TRANQUILO?

```
public int metodo(int a, double b) {...}

public void metodo(int a, long b) {...}
// OK - tipos dos parâmetros diferentes

public long metodo(int x, double b) {...}
// Erro - é igual a 1ª definição

public void metodo(int a, long b, float c) {...}
// OK - número de parâmetros diferente

public void metodo(double a, int b) {...}
// OK - ordem de parâmetros diferente
```

Prof. Marcos Vinicius – UFC/Russas - POO

28/50

## VAMOS FALAR UM POUCO SOBRE CONSTRUTORES

- **Construtores** são “métodos” especiais chamados no momento da criação de um objeto.
- Servem para **inicializar** objetos de forma organizada, ou seja, servem para “**setar**” o estado inicial de um objeto quando ele é criado.
- Pode haver mais de um construtor por classe (**overloading** ou sobrecarga de métodos).

Prof. Marcos Vinicius – UFC/Russas - POO

29/50

## CONSTRUTORES: EXEMPLO 1

```
public class Data {  
    int dia, mes, ano;  
  
    public Data() {  
        ano = 2010; }  
    ...  
}
```

**Utilizando...**

```
...  
Data hoje = new Data();
```

Eu já usava  
e não  
sabia!



Prof. Marcos Vinicius – UFC/Russas - POO

30/50

## RESTRIÇÕES DOS CONSTRUTORES

- Possuem algumas restrições:
  - ✓ devem ter o mesmo nome da classe;
  - ✓ não possuem valor de retorno (**nem mesmo void**);
  - ✓ podem ter modificadores de acessibilidade (public, ...);
  - ✓ são chamados quando o operador **new** é executado.



Prof. Marcos Vinicius – UFC/Russas - POO

31/50

## CONSTRUTORES: EXEMPLO 2

```
public class Data {
...
    public Data() {
        ano = 2004; }
    public Data( int numAnos ) {
        ano = numAnos; }
...
}
```

Utilizando...

```
...
Data hoje = new Data();
Data depois = new Data( 2005 );
```

Prof. Marcos Vinicius – UFC/Russas - POO

32/50



## CONSTRUTOR THIS()

- Construtores podem ser sobrecarregados, mas **somente na mesma classe**.
- A chamada do construtor **this ()** pode ser usada para encadear construtores sobrecarregados (**não confundir com a referência this**).

Estava fácil demais!



Prof. Marcos Vinicius – UFC/Russas - POO

33/50

## CONSTRUTOR THIS()

- A chamada **this ()** só pode ser usada em definições de construtores, e, quando usada, **deve sempre ser a primeira sentença no código do construtor**

```
public Estudante(String nome, char
sexo) {
    this.sexo = sexo;
    this.nome = nome; }
```

```
public Estudante(String nome, char
sexo, int matricula){
    this( nome, sexo );
    this.matricula = matricula; }
```

Prof. Marcos Vinicius – UFC/Russas - POO

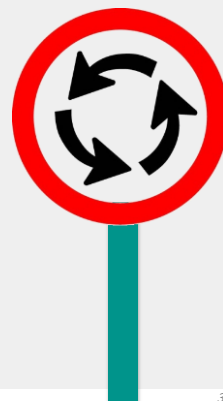
34/50



# Iterações

## INTRODUÇÃO

- As estruturas de controle de iteração fazem com que determinadas ações sejam repetidas até que uma determinada condição seja satisfeita.
- Em Java temos:
  - ✓ **while**
  - ✓ **do...while**
  - ✓ **for**

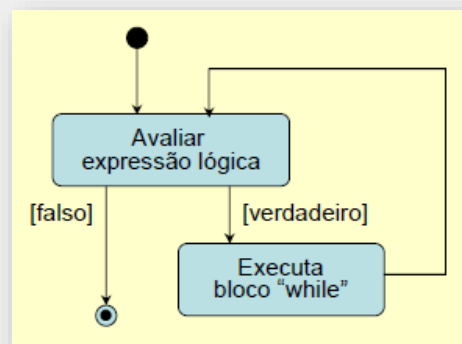


## ITERAÇÃO COM TESTE NO INÍCIO: WHILE

- Neste tipo de estrutura a condição de iteração é avaliada antes da execução do bloco do laço.

```
while ( <condição> ) {
    <bloco do laço>
}
```

```
while ((x>y) && test()) {
    façaAlgo();
    façaOutraCoisa();
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

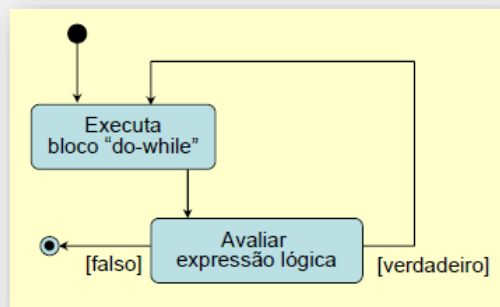
37/50

## REPETIÇÃO COM TESTE NO FINAL: DO... WHILE

- Neste tipo de estrutura a condição de iteração é avaliada depois da primeira execução do bloco do laço.

```
do {
    <bloco do laço>
} while ( <condição> );
```

```
do {
    façaAlgo();
    façaOutraCoisa();
} while ((x > y) && test());
```



Prof. Marcos Vinicius – UFC/Russas - POO

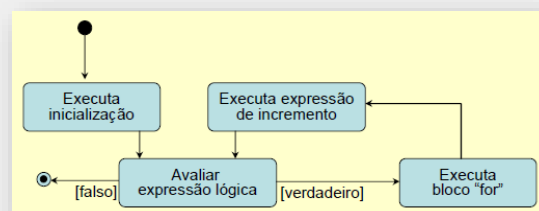
38/50

## ITERAÇÃO FACILITADA: FOR

- É o laço de repetição onde a inicialização, a expressão para determinação do passo de iteração e a condição de parada são definidos em um único local!

```
for(<inicializações>;<condição>;<expressão inc/dec>){
    <bloco do laço>
}
```

```
for( int i = 0 ; i < 10 ; i++ ){
    facaAlgo();
    facaOutraCoisa();
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

39/50



## Estruturas de Transferência

## INTRODUÇÃO

- Em Java temos **três estruturas de transferência**:

`break`

`continue`

`return`



Prof. Marcos Vinicius – UFC/Russas - POO

41/50

## BREAK

- O **break** pode ser usado em blocos com rótulos, laços de repetição e no **switch**, para transferir o fluxo para fora do contexto corrente, ou seja, para o bloco mais externo.

```
for ( i=0; i<5 ; i++) {
    if ( i == 3 ) {
        break;
    }
    System.out.println( "i = " + i );
}
```



Voadora bem na  
pleura central da  
peridural!




Prof. Marcos Vinicius – UFC/Russas - POO

42/50

## O QUE ACONTECE NO TRECHO ABAIXO?

```
for ( i=0; i<4 ; i++) {
    if ( i == 2 )
        break;
    for ( j=0; j<4; j++) {
        if ( j == 2 )
            break;
        System.out.println( "j = " + j );
    }
    System.out.println( "i = " + i );
}
```



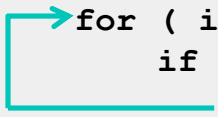

Prof. Marcos Vinicius – UFC/Russas - POO

43/50

## CONTINUE

- O **continue** pode ser usado em laços de repetição (**for**, **while**, e **do-while**), para interromper prematuramente o fluxo de execução e **avançando para a próxima iteração**.

```
for ( i=0; i<5 ; i++) {
    if ( i == 3 ) {
        continue;
    }
    System.out.println( "i = " + i );
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

44/50

## RETURN

- O **return** é usado para parar a execução do método corrente e transferir o controle para o método que o chamou.

```
private void calc() {  
    for ( i=0; i<5 ; i++) {  
        if ( i == 3 )  
            return;  
    }  
    ...  
    calc ();  
}
```

```
private int somar(int a, int b){  
    int s = 0;  
    s = a + b;  
    return s;  
}  
...  
int x = somar( 3, 4 );
```

Prof. Marcos Vinicius – UFC/Russas - POO

45/50





Utilize controles de fluxo nos métodos da classe **Bovino**.

Faça que somente datas válidas sejam aceitas.

Crie um método que informe se um ano é bissexto.

Vamos **brincar** mais um pouco?





**Revise os métodos** da classe **Robot** e faça uso de **controle de fluxo**.

Insira os métodos de **getters** e **setters**!



**Obrigado!**  
**Mais alguma dúvida?**

Acesse o **AME** para mais informações e treinamento do **NERDS**!

<http://ame2.russas.ufc.br>

