



PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Marcos Vinicius de Andrade Lima
E-mail: marcos.vinicius@ufc.br



Olá!

Sou Marcos Vinicius

No tópico passado nós aprendemos como inserir características nos objetos...

Neste tópico começaremos a **definir** o **comportamento** dos nossos objetos!

“

**Se você não sabe onde quer ir,
qualquer caminho serve**
(Lewis Carroll)



**Definindo Comportamentos
dos Objetos**

MÉTODOS

- O dicionário Aurélio define método como o caminho pelo qual se atinge um objetivo.
- Então, método é a característica que possibilita alterar a funcionalidade de um atributo.
- Usando uma definição mais ampla pode-se conceituar método como o controle lógico que reflete uma ação (designa comportamento) sobre o objeto.

MÉTODOS EM JAVA

- **Um método em Java no mínimo deve ter:**
 - ✓ um modificador de acesso (visibilidade)
 - ✓ um ou nenhum tipo de retorno
 - ✓ um nome
 - ✓ nenhum, um ou vários parâmetros
- **Cada parâmetro pode receber como argumento:**
 - ✓ um valor
 - ✓ uma variável
 - ✓ uma expressão
- **O tipo de retorno pode ser:**
 - ✓ qualquer tipo primitivo (int, long, float, ...)
 - ✓ qualquer classe (String, ...)
 - ✓ void, para o caso de não retornar nada (procedimento)

MÉTODOS: EXEMPLO

```
public class DeclaracaoDeMetodos {
// declarando um método (que representa uma função...)
    public int soma ( int a , int b ) {
        int s = a + b;
        return s;
    }

// declarando um método que representa
// um procedimento...
    public void imprimir ( int a ) {
        System.out.println ( a );
    }
}
```

Observe os
tipos de
retorno...



Prof. Marcos Vinicius – UFC/Russas - POO

7/50

UM TOQUE ESPECIAL PARA GALERA DO C

- Procedimentos e funções são representados em POO por **métodos**.
- A diferença está no tipo de retorno. Se o tipo de retorno for nulo, indicado pela palavra **void**, o método representa um procedimento. Caso contrário (se existe um tipo de retorno), o método representa uma função.



Prof. Marcos Vinicius – UFC/Russas - POO

8/50

É BOM SABER...

- Qual a diferença entre parâmetro e argumento?
 - **parâmetro** representa um **valor que o método espera** que você passe quando você chamá-lo;
 - **argumento** representa o **valor que você passa** para um parâmetro de método, quando você chama o método.

Só peço um dia de paz para minha mente!



Prof. Marcos Vinicius – UFC/Russas - POO

9/50

VEJA BEM...

```
// executando o método
Operacao op = new Operacao();
int x = 10;
int z = op.soma ( x , 20 );
```

```
...
// declarando o método em Operacao
public int soma (int a , int b )
{
    int s = a + b;
    return s;
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

10/50

OBSERVAÇÕES IMPORTANTES

- O **número de argumentos** passados deve ser **igual ao da definição**.
- **Cada argumento** individualmente, deve ter **tipo compatível** ao da definição.
- Para **tipos primitivos**, a passagem é **por valor**, ou seja, o valor (e não o endereço) da variável é passado.
- **Cada chamada** a um **método** cria suas próprias **instâncias dos parâmetros formais**.

Prof. Marcos Vinicius – UFC/Russas - POO

11/50



CUIDADO!!!

Em Java, **todos** os argumentos são passados por valor, isto significa que o valor (e não o endereço) da variável é passado. Assim, as modificações no parâmetro formal (dentro do método) não afetam os valores das variáveis passadas nos argumentos (valor externo).

ACESSIBILIDADE PÚBLICA E PRIVADA

- 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

13/50

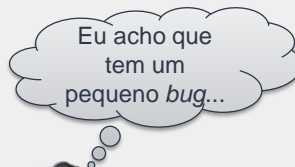
Não é recomendável deixar que os **outros sejam responsáveis** pelas **validações das variáveis de instâncias!**

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



UM EXEMPLO A SER EVITADO...

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



Prof. Marcos Vinicius – UFC/Russas - POO

15/50

Para evitar que situações como essa 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

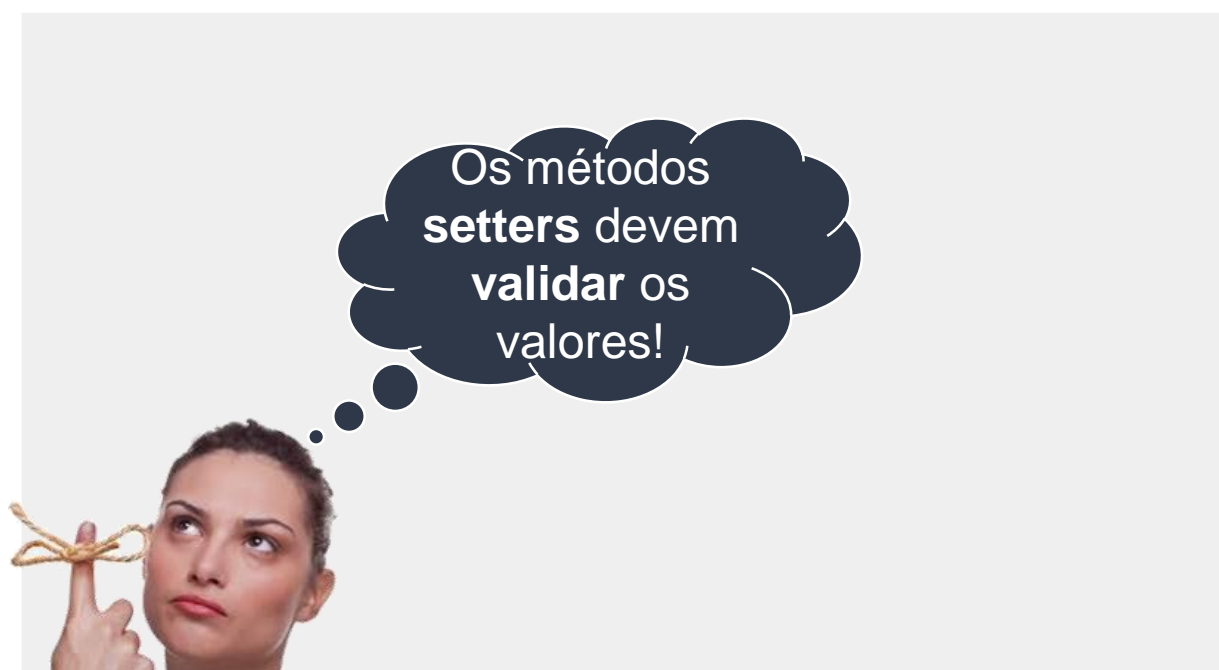
17/50

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

18/50



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;
```

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

21/50

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

22/50

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

23/50

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

24/50

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

25/50

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

26/50

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

27/50

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

28/50

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

29/50

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

30/50



Pacotes em Java

ORGANIZANDO AS CLASSES EM PACOTES

- Um pacote serve para agrupar classes, interfaces e subpacotes inter-relacionados
- Cada pacote está associado a um diretório do Sistema Operacional, mas nem todo diretório é um pacote
- Classes de pacotes diferentes podem ter o mesmo nome e que o uso de pacotes influencia na acessibilidade dos objetos
- **ATENÇÃO:** pacotes estão associados ao agrupamento de arquivos “.class” e não de arquivos “.java”!

DEFININDO UM PACOTE EM JAVA

- Para declarar um determinado pacote, usa-se a cláusula **package** no início do arquivo:

`package faculdade;`
- Supondo que o diretório **C:\home** está no **CLASSPATH**, as classes do pacote **faculdade** devem estar no diretório **C:\home\faculdade**
- Em cada arquivo fonte desse mesmo pacote deve haver a mesma cláusula **package** no início

Prof. Marcos Vinicius – UFC/Russas - POO

33/50

DEFININDO UM SUBPACOTE EM JAVA

- Para definirmos subpacotes basta fazermos uso do operador ponto “.”
- `package faculdade.classes;`
- Supondo que o diretório **C:\home** está no **CLASSPATH**, as classes do pacote **faculdade.classes** devem estar no diretório **C:\home\faculdade\classes**
- Em cada arquivo fonte desse mesmo pacote deve haver a mesma cláusula **package** no início

Prof. Marcos Vinicius – UFC/Russas - POO

34/50

OLHA SÓ QUE LEGAL!

- Classes contidas em pacotes são identificadas prefixando-se o nome do pacote ao seu nome:

faculdade.classes.Estudante

(classe Estudante contida no pacote faculdade.classes)

java.awt.Button

(classe Button faz parte do pacote java.awt)

java.sql.Date

java.util.Date

Embora as classes possuam o mesmo nome, elas são diferentes!

UTILIZANDO CLASSES EM PACOTES

- As classes de um pacote podem ser utilizadas por outra classe da seguinte maneira:
 - ✓ usando o nome da classe totalmente qualificado (especificando todo o caminho)
 - ✓ a partir de um **import** usando o operador ponto (.)

- Exemplos:**

```
class TesteEstudante{
    faculdade.classes.Estudante e;
    ...
}
```

```
import faculdade.classes.Estudante;
class TesteEstudante{
    Estudante e;
    ...
}
```

Pode-se ainda usar o asterisco (*) para importar todas as classes de um mesmo pacote:

```
import faculdade.classes.*;
```

ATENÇÃO: o asterisco (*) não importa subpacotes, somente as classes daquele pacote! Podem existir várias importações em um mesmo arquivo

NOMEANDO PACOTES

- Há um esquema global de nomes para pacotes, que segue o padrão de nomes de domínio da internet, escrevendo a URL das empresas de trás para frente, o que permite identificação única de classes
- O objetivo é evitar conflito entre classes de fornecedores diferentes!
- **Exemplo:**

```
br.ufc.russas.nerds
```


ACESSIBILIDADE DE PACOTE EM JAVA

- Além das acessibilidades pública (public) e privada (private), há uma outra associada aos pacotes
- É a acessibilidade padrão (default), que indica que somente classes dentro de um mesmo pacote podem ter acesso aos métodos e variáveis
- Em **Java** esse modificador é **utilizado quando não se coloca nada** em classes, métodos ou variáveis!
- **Exemplos:**

```
long tamanho;  
long getTamanho() { return tamanho; }
```

Prof. Marcos Vinicius – UFC/Russas - POO

39/50



Depois não diga
que eu não avisei!

Para **classes** e **interfaces** em Java só existem **dois** tipos de acessibilidade: **pública** ou **pacote**.



VAMOS PENSAR UM POUCO...

O que um bovino faz?



Olha eu aqui outra vez!



Prof. Marcos Vinicius – UFC/Russas - POO

43/50

AÇÕES DE OBJETOS BOVINOS

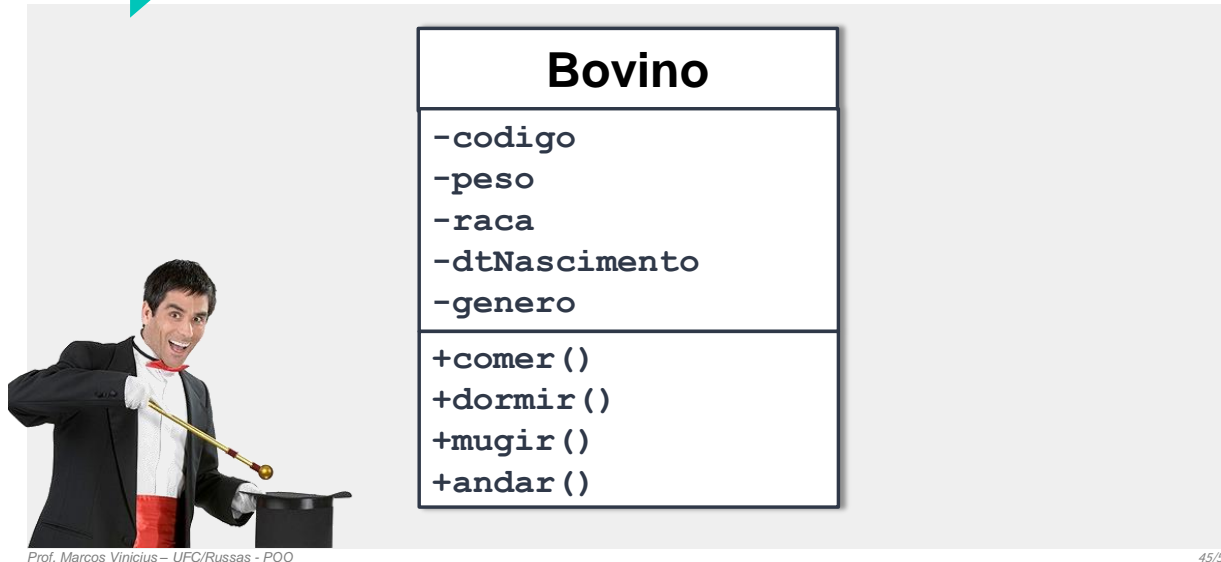
- Comer → Todo bovino deve se alimentar
- Dormir → Todo bovino deve dormir
- Mugir → Todo bovino produz o som “muuu”
- Abanar o Rabo → Todo bovino deve abanar o rabo
- Andar → Todo bovino deve andar pelo pasto.



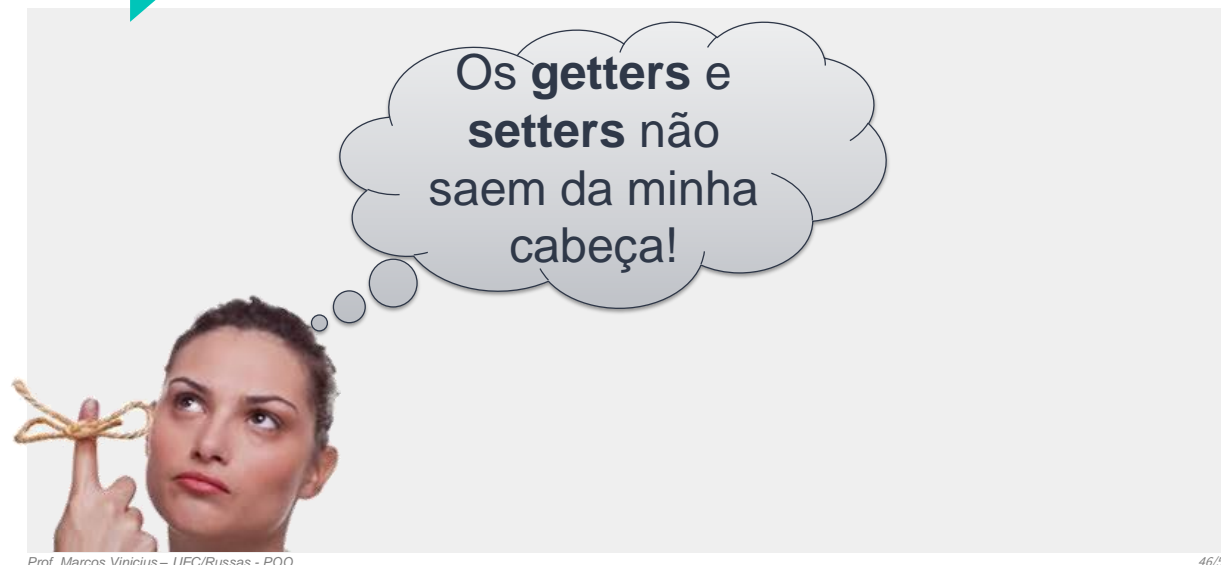
Prof. Marcos Vinicius – UFC/Russas - POO

44/50

CLASSE BOVINO EM UML



NÃO SE ESQUEÇA DOS MÉTODOS ACESSADORES E MODIFICADORES!



CRIE AO MENOS DOIS CONSTRUTORES!

PLEASE... Crie construtores úteis!



Prof. Marcos Vinicius – UFC/Russas - POO

47/50

Vamos **brincar** mais um pouco?





Os objetivos da **classe Robot** agora **devem guardar sua localização no plano cartesiano**. O que deverá ser modificado na classe para que isto seja possível?

Adicione **construtores** e **movimentação** ao robô.

Como é realizada uma entrada de dados via teclado em Java? (Dica: pesquise sobre **Scanner()**)



Obrigado!
Mais alguma dúvida?

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

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

