



Programação Orientada a Objetos – Aula 05

IFPE – Campus Igarassu
2016.1

Ranieri Valença
28/07/2016



Tópicos de hoje

- Revisão
 - Métodos com retorno e parâmetros
 - Parâmetros X argumentos
 - Passagem por valor vs. Passagem por referência
- Construtores
- this



Lembrando...

Dizemos que um objeto é uma
Instância
de uma classe



Lembrando...

Para instanciar um objeto em Java
(e em algumas outras linguagens – tipo PHP)

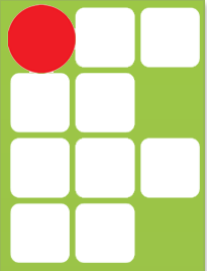
Usando o operador **new**



Comportamentos dos objetos

No mundo real, os objetos têm
ações e comportamentos

Essas ações e comportamentos são
comuns a todos os **objetos** similares –
ou seja, comuns à **classe**



Traduzindo isso para nosso mundo...

Comportamentos de objetos no mundo real são traduzidos em **Métodos** no paradigma orientado a objetos



Lembrando o que é um método

São **subprogramas** que contém
instruções de execução

A ideia é que cada método execute
uma única tarefa

Onde ficam os métodos?? Dentro da classe!

```
class Pessoa {
```

```
    int idade;  
    float pesoEmQuilogramas;  
    float alturaEmMetros;  
    char sexo;  
    String nome;  
    String sobrenome;  
    String[] telefone;
```

Atributos

```
    void imprimeNome() {  
        System.out.println("O indivíduo se chama "  
            + nome);  
    }
```

```
}
```

Métodos

Invocando métodos em Java

```
class App {  
    public static void main(String[] args) {  
        Pessoa p;  
        p = new Pessoa();  
        p.idade = 30;  
        p.nome = "Tom Riddle";  
  
        p.imprimeNomeEIdade();  
    }  
}
```

Operador “.” (ponto)

Invocando métodos em Java

```
class App {  
    public static
```

Métodos são invocados a partir
de **objetos** (instâncias)
(por enquanto)

```
        Tom Riddle";  
    }
```

```
        p.imprimeNomeEIdade();  
    }
```

Operador “.” (ponto)



Métodos que retornam algo

Utilizamos a palavra especial “return” para que um método retorne algo

Obviamente quando um método retorna algum valor, seu **tipo de retorno** precisa ser especificado



Métodos que retornam algo

```
int alturaEmCentimetros() {  
    return alturaEmMetros * 100;  
}
```

```
int altura = p.alturaEmCentimetros();  
System.out.println("A criatura tem " + altura + "cm");
```



Métodos parametrizados

Podemos criar métodos que **recebem um**
ou mais parâmetros, e que seu
comportamento, assim como seu retorno,
depende desse(s) parâmetro(s)

Métodos parametrizados

```
class Numero {  
    boolean ehDivisivel(int a, int b) {  
        if (a % b == 0) {  
            return true;  
        }  
        return false;  
    }  
}
```

↑
Vírgula para
separar os
parâmetros



Métodos parametrizados

```
class App {  
    public static void main(String[] args) {  
        Numero n = new Numero();  
        System.out.println(n.ehDivisivel(7, 3));  
        System.out.println(n.ehDivisivel(28, 4));  
    }  
}
```



Parâmetro vs Argumento – A batalha épica

Argumentos são os valores passados para um método durante sua **invocação**

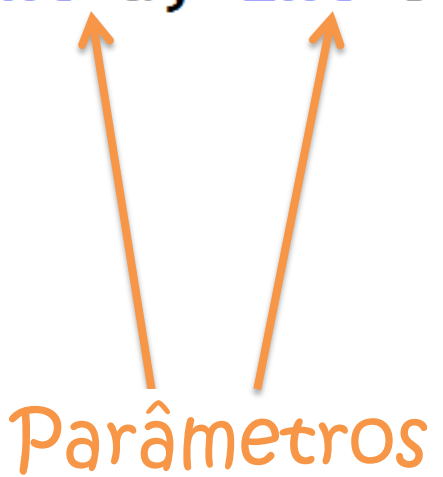


Parâmetro vs Argumento – A batalha épica

Parâmetros são as variáveis declaradas no método, que servem apenas no **escopo** daquele método e que recebem os valores dos argumentos

Parâmetro vs Argumento – A batalha épica

```
class Numero {  
    boolean ehDivisivel(int a, int b) {  
        if (a % b == 0) {  
            return true;  
        }  
        return false;  
    }  
}
```



Parâmetros

Parâmetro vs Argumento – A batalha épica

```
class App {  
    public static void main(String[] args) {  
        Numero n = new Numero();  
        System.out.println(n.ehDivisivel(7, 3));  
        System.out.println(n.ehDivisivel(28, 4));  
    }  
}
```

Argumentos



Argumentos





Prática 1

1. Crie um método chamado **“combustivelNecessario”** na classe **“Veiculo”**, que recebe um argumento que corresponde ao numero de km para o qual se deseja calcular a quantidade de combustível necessária
2. Crie um método chamado **“dinheiroNecessario”** na classe **“Veiculo”** que recebe como argumentos a quantidade de km e o preço do litro de combustível



Referências e valores

As linguagens orientadas a objetos possuem
o conceito de **referência**

Uma atribuição não necessariamente cria
uma **cópia**

~~Como é pae?~~

Referências e valores

*“y” recebe o
valor de “x”*

```
int x = 7;  
int y;
```

```
y = x;
```

```
y = 12;
```

```
System.out.println(x);
```

```
System.out.println(y);
```

*O valor de “y” é
alterado, mas não
o de “x”*

Referências e valores

```
Pessoa anakin;
```

```
Pessoa vader;
```

```
anakin = new Pessoa();  
anakin.nome = "Han Solo";
```

```
vader = anakin;  
vader.nome = "Lord Vader";
```

```
System.out.println(anakin.nome);  
System.out.println(vader.nome);
```

*anakin é uma nova instância
e seu nome foi alterado*


*vader recebe uma
referência de
anakin*



Prática 2

1. Declare duas variáveis do tipo “Veiculo”
 - `veiculo1` e `veiculo2`
2. Inicialize `veiculo1`
3. Faça `veiculo2 = veiculo1;`
4. Altere os valores dos atributos de `veiculo1`
5. Imprima (`System.out.println`) os atributos de `veiculo2`

O que acontece?

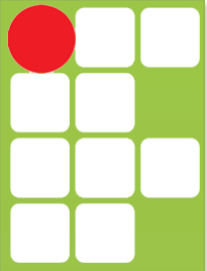


Continuando na quest para dominar as Classes...

Estamos perto de dominar as classes.

Uma vez dominadas, poderemos aplicar orientação a objetos em tantos lugares...

Tudo será muito mais reutilizável, “ctrl+c” (copiar) e “ctrl+v” (colar) terão inveja de nossa capacidade de escapar de suas garras...

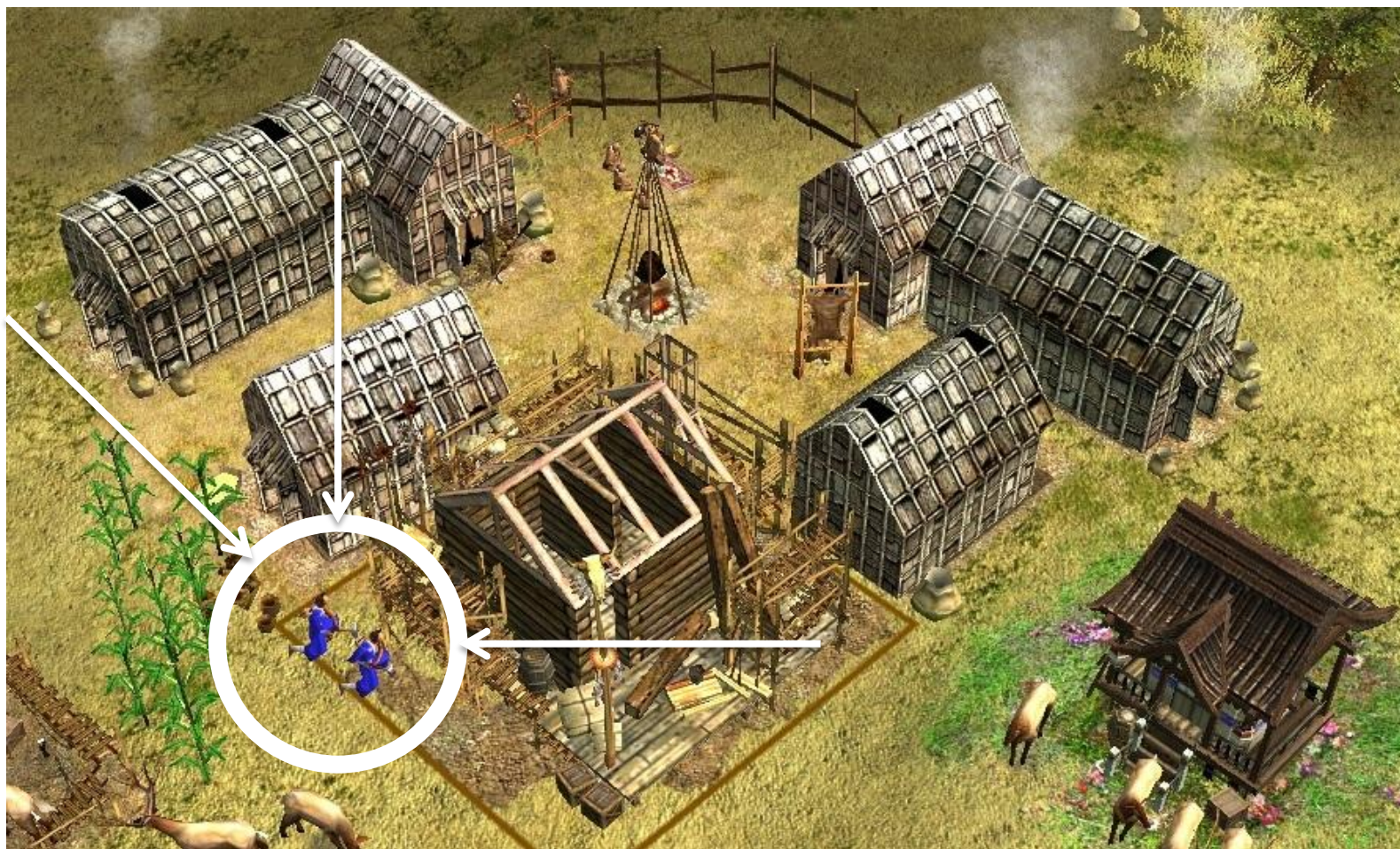


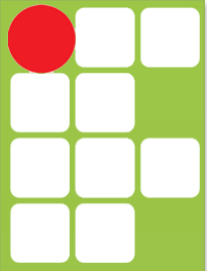
Continuando na quest para dominar as Classes...

Agora nos faltam apenas alguns poucos conceitos importantes.

E aqui vai um deles: **construtores**

Construtores





Construtores

O que acontece quando chamamos o
operador **new**?



Construtores

O que acontece quando chamamos o operador
new?

Se não houver construtor explícito ~~(+i)~~, o Java
invoca o método **construtor** padrão
da classe



Construtores

Até agora, temos criado os objetos usando apenas
“new nome-da-classe();”

E depois alteramos seus atributos um a um. Mas
isso é ruim. MUITO RUIM.

Veiculo tem seis atributos. É fácil de se perder
entre esses atributos, trocar os nomes das
variáveis, esquecer um atributo, ...



Construtores

Veiculo tem seis atributos. É fácil de se perder entre esses atributos, trocar os nomes das variáveis, esquecer um atributo, ...

Tudo isso porque estamos usando o construtor padrão, que não recebe nenhum argumento e apenas inicializa o objeto “Veiculo”.



Exemplo de Construtor

```
class Pessoa {  
    int idade;  
    float pesoEmQuilogramas;  
    float alturaEmMetros;  
    char sexo;  
    String nome;  
    String sobrenome;  
    String[] telefone;
```

```
Pessoa() {  
    telefone = new String[10];  
}
```


Exemplo de Construtor

```
class Pessoa {  
    int idade;  
    float pesoEmQuilogramas;  
    float alturaEmMetros;  
    char sexo;  
    String nome;  
    String sobrenome;  
    String[] telefone;  
}
```

Mesmo nome da Classe

```
Pessoa() {  
    telefone = new String[10];  
}
```

Construtores não têm tipo de retorno!!!

Exemplo de Construtor parametrizado

```
class Pessoa {  
    int idade;  
    float pesoEmQuilogramas;  
    float alturaEmMetros;  
    char sexo;  
    String nome;  
    String sobrenome;  
    String[] telefone;  
}
```

Construtores
podem ter
parâmetros



```
Pessoa(String n, String s, int i) {  
    nome = n;  
    sobrenome = s;  
    idade = i;  
    telefone = new String[10];  
}
```

Exemplo de Construtor parametrizado

Argumentos para o construtor

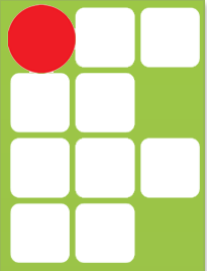
```
Pessoa p = new Pessoa("Luke", "Skywalker", 18);
```





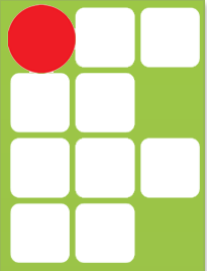
Prática 3

1. Crie um construtor para a classe “Veiculo” que recebe os seis parâmetros necessários para inicializar seus atributos
2. Altere as chamadas no método “main” para utilizarem o novo construtor



this

Todo método que é invocado numa classe
recebe um argumento especial: **this**



this

O “this” refere-se ao objeto no qual o método está sendo chamado

~~0i?~~



this

```
int alturaEmCentimetros() {  
    return this.alturaEmMetros * 100;  
}
```

```
Pessoa p;  
p = new Pessoa();
```

```
int altura = p.alturaEmCentimetros();  
System.out.println("A criatura tem " + altura + "cm");
```

this

```
int alturaEmCentimetros() {  
    return this.alturaEmMetros * 100;  
}
```

“this” refere-se a “p”, porque o método está sendo chamado a partir do objeto “p”

“p” é um objeto do tipo Pessoa

```
Pessoa p;  
p = new Pessoa();
```

Invocamos o método do objeto “p”

```
int altura = p.alturaEmCentimetros();  
System.out.println("A criatura tem " + altura + "cm");
```




this

E como usamos o *this*?



Exemplo de this num construtor

```
Pessoa(String n, String s, int i) {  
    this.nome = n;  
    this.sobrenome = s;  
    this.idade = i;  
    this.telefone = new String[10];  
}
```



Exemplo de this num construtor

```
Pessoa(String nome, String sobrenome, int idade) {  
    this.nome      = nome;  
    this.sobrenome = sobrenome;  
    this.idade     = idade;  
    this.telefone  = new String[10];  
}
```

Exemplo de this num construtor

Parâmetros

```
Pessoa(String nome, String sobrenome, int idade) {  
    this.nome = nome;  
    this.sobrenome = sobrenome;  
    this.idade = idade;  
    this.telefone = new String[10];  
}
```

Atributos



Prática 4

1. Altere os métodos da classe “Veiculo” para utilizarem o “this” ao referir-se aos atributos do objeto

