# Programação Orientada a Objetos (PO24CP)

**Aula #08 - Sobrecarga de métodos e palavras reservadas: final, this e static**

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

# Sobrecarga de métodos

# Sobrecarga de métodos

## Métodos sobrecarregados devem possuir assinaturas diferentes

Neste caso, a assinatura de método é representada pelo número de parâmetros e pelo tipo dos parâmetros

```
1  public class Data{
2   private int dia, mes, ano;
3
4   public void alterarData(int d){
5      this.dia = d;
6   }
7   public void alterarData(int d, int m){
8      this.dia = d; this.mes = m;
9   }
10  public void alterarData(int d, int m, int a){
11     this.dia = d; this.mes = m; this.ano = a;
12  }
13 }
```

```
14 Data d = new Data();
15 d.alterarData(31);
16 d.alterarData(31,12);
17 d.alterarData(31,12,1969);
```

# Palavra reservada: this

# Palavra reservada: `this` (referência ao *objeto corrente*)

```
18 public class Complexo{
19  private int real;
20  private int imaginario;
21
22  public Complexo(int real, int imaginario){
23      this.real = real; this.imaginario = imaginario;
24  }
25  public void soma(Complexo c){
26     this.real = this.real + c.real;
27     this.imaginario = this.imaginario + c.imaginario;
28  }
29  public void imprimir(){
30     System.out.println(this.real + "," + this.imaginario);
31  }
32 }
```

```
33 Complexo a = new Complexo(1,2);
34 Complexo b = new Complexo(3,4);
35 a.soma(b);
36 a.imprimir();//4,6
37 b.imprimir();//3,4
```

# Palavra reservada: `this` (referência ao *objeto corrente*)

```
38 public class Bicicleta{
39  private double valor;
40
41  public void imprimirValor(){
42   System.out.println(this.
         valor);
43  }
44 }
```

```
45 public class Principal{
46 private double valor;
47
48 public void imprimirValor(){
49  System.out.println(this.valor);
50 }
51 public void teste(){
52  Bicicleta b = new Bicicleta();
53  b.imprimirValor();
54 }
55 public static void main(String[]
       args){
56  Principal p = new Principal();
57  p.imprimirValor();
58  p.teste();
59  }
60 }
```

1. Na linha 49 o `this` é referência para o objeto de qual classe? `Principal` ou `Bicicleta`?
2. A linha 58 poderia ser substituída por `this.teste()`?

# Palavra reservada: this (auto referência do objeto)

```java
public class Pessoa{
 private String nome;
 private String cpf;

 public Pessoa(String nome,String cpf){
  this.nome = nome; this.cpf = cpf;
 }
 public String toString(){
  return this.nome+", "+this.cpf;
 }
 public Pessoa getPessoa(){
  return this;// retornando sua própria referência
 }
}
```

```java
public class UsaPessoa{
 public static void main(String[] args){
  Pessoa fulano = new Pessoa("Fulano da Silva","543");
  Pessoa ciclano = fulano.getPessoa();
  System.out.println(fulano);  //Fulano da Silva, 543
  System.out.println(ciclano); //Fulano da Silva, 543
 }
}
```

# Usando this como um Construtor

## Invocação de construtor explícita

De dentro de um construtor, pode-se usar `this` para chamar outro construtor na mesma classe.

```
83  public class Rectangle {
84      private int x, y;
85      private int width, height;
86
87      public Rectangle() {
88          this(0, 0, 1, 1);
89      }
90      public Rectangle(int width, int height) {
91          this(0, 0, width, height);
92      }
93      public Rectangle(int x, int y, int width, int height) {
94          this.x = x;
95          this.y = y;
96          this.width = width;
97          this.height = height;
98      }
99      ...
100 }
```

# Membros de classe estáticos: palavra reservada static

# Membros de classe estáticos: `static`

## Atributos **não estáticos**

Cada instância da classe terá uma **cópia distinta** deste atributo.

```
101 public class Celular{
102  private int total;
103  public Celular(){
104   this.total = this.total + 1;
105  }
106  public void incrementar(){
107   this.total = this.total + 1;
108  }
109  public int getTotal(){
110   return this.total;
111  }
112 }
```

```
113 Celular a = new Celular();
114 Celular b = new Celular();
115 a.incrementar(); b.incrementar();
116
117 System.out.println(a.getTotal()); // o que sera' impresso?
118 System.out.println(b.getTotal()); // o que sera' impresso?
```

# Membros de classe estáticos: `static`

## Atributos **estáticos**

ficam **comuns para todos** os objetos que foram instanciados para esta classe, sendo assim chamados de "atributos da classe"

- Não se pode usar o `this` para acessar um membro estático. Deve-se usar o nome da **Classe**

## Métodos **estáticos**

Classes podem possuir métodos estáticos e estes podem ser invocados sem que necessite criar uma instância da classe

- Métodos estáticos geralmente são usados para acessar atributos estáticos

```
119 public class Celular{
120  private static int total = 0;
121  public static int getTotal(){
122     return Celular.total;
123  }
124 }
```

# Membros de classe estáticos: static

```
125 public class Celular{
126   private static int total = 0;
127   private int serial;
128
129   public Celular(int s){
130     this.serial = s;
131     Celular.total = Celular.total + 1;
132   }
133   public static int getTotal(){
134     return Celular.total;
135   }
136   public int getSerial(){
137     return this.serial;
138   }
139 }
```

```
140 System.out.println(Celular.getTotal()); // o que sera' impresso?
141 Celular c = new Celular(123);
142 Celular d = new Celular(456);
143 System.out.println(Celular.getTotal()); // o que sera' impresso?
144 System.out.println(d.getSerial()); // o que sera' impresso?
```

# Constantes com o modificador final

# Constantes com o modificador `final`

## O modificador **final**

pode ser usado em atributos ou métodos de uma classe, bem como em variáveis locais

- Uma vez que atribuiu valores para variáveis ou atributos, estes não poderão ser alterados
- Por convenção, constantes deverão ser escritas em letras maiúsculas
- Métodos não poderão ser sobrescritos (conceito de herança)

```
145 public class Celular{
146   private final int FREQUENCIA = 1800;
147   private final int SERIAL;
148
149   public Celular(int s){
150     this.SERIAL = s;    }
151
152   public final void iniciarChamada(){
153    /* ...  */  }
154 }
```

# Resumo

- **Sobrecarga de métodos**
  - Uma classe pode ter mais de um método com o mesmo nome, porém com assinaturas diferentes
- **A palavra `this` é uma referência para o objeto atual**
  - Apesar de não obrigatório na maioria dos casos, seu uso é desejado para facilitar a leitura do código
- **Atributos estáticos ficam comum para todos objetos instanciados da classe**
  - Imagine que é uma variável compartilhada entre todos os objetos da classe
- **Modificador `final` é usado para definir constantes**
  - Atributos `final` não poderão ter seu valor alterado
  - Métodos `final` não poderão ser sobrescritos (herança)

# Exercício 1

Analise a classe **utilitária** Math (java.lang.Math) e as afirmativas abaixo:

- http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

1. Para obter a raiz quadrada do número 4, basta: double d = Math.sqrt(4);
2. Math.PI é uma constante que contém o valor aproximado de PI. Para imprimir este valor, basta: System.out.println(Math.PI);

**Responda**

Dos conceitos apresentados nesta aula, quais deles a classe `java.lang.Math` faz uso? Justifique sua resposta

# Exercício 2

1. Crie uma classe utilitária para trabalhar com datas. A classe deverá prover as seguintes funcionalidades
   - Receber uma data como parâmetro e retornar uma String com a data por extenso
     - Se receber somente um inteiro, então retornar o dia por extenso
     - Se receber dois inteiros, então retornar dia e mês por extenso
     - Se receber três inteiros, então retornar dia e mês por extenso e ano (não precisa ser por extenso)
   - Receber um inteiro como parâmetro e retornar o nome do respectivo mês. P.e., ao passar o número 2, a classe deve retornar "fevereiro"
   - Receber duas datas como parâmetro no formato (dia, mês, ano) e retornar a diferença em dias entre estas (primeira - segunda)
   - Indicar se o ano recebido como parâmetro é ou não um ano bissexto.
     - Bissexto = todo ano divisível por 400 ou (divisível por 4, porém não divisível por 100)
2. Escreva um aplicativo Java e invoque os métodos que criou na classe utilitária Data

# Exercícios extras

- Faça os exercícios de 1 a 4 da seção 7.7 (página 94) da apostila da Caelum
  - Disponível no moodle da disciplina
- Material de referência
  - http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
  - http://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html
  - http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html