Métodos em Java

Curso de Linguagem Java

José Antonio F. de Macêdo
jose.macedo@lia.ufc.br

Métodos

- Define o comportamento da classe.
- Declaração:

```
class Ponto {
  int x, y;

  void mover ( int dx, int dy) {
    x += dx;
    y += dy;
  }
}
```

Tipos de Métodos

Métodos Construtores

Método Operacional

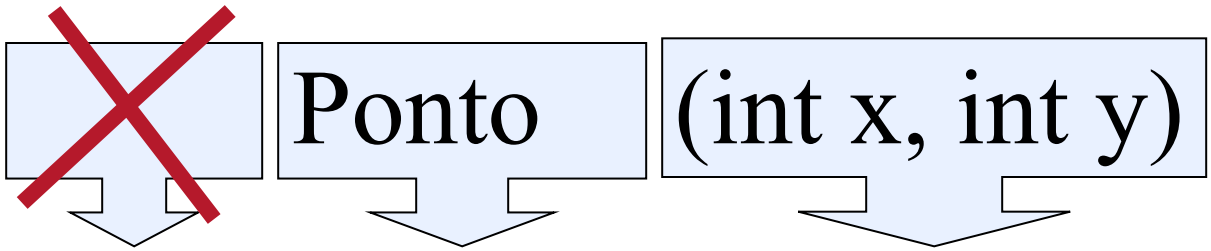
```
class Ponto
{
   int a;
    int x, y;

    Ponto()
    {
        …
    }

   void mover ( int dx, int dy)
   {
     x += dx;
     y += dy;
   }
}
```

Métodos Construtores

- Devemos usar os métodos construtores quando queremos atribuir valores aos atributos de um objeto no momento de sua criação

```
class Ponto {
    int x=0;
    int y=0;
    Ponto (int x, int y)
    {
          this.x = x;
          this.y = y;
     }
}
```

Assinatura de um método Construtor



Não possui tipo de retorno

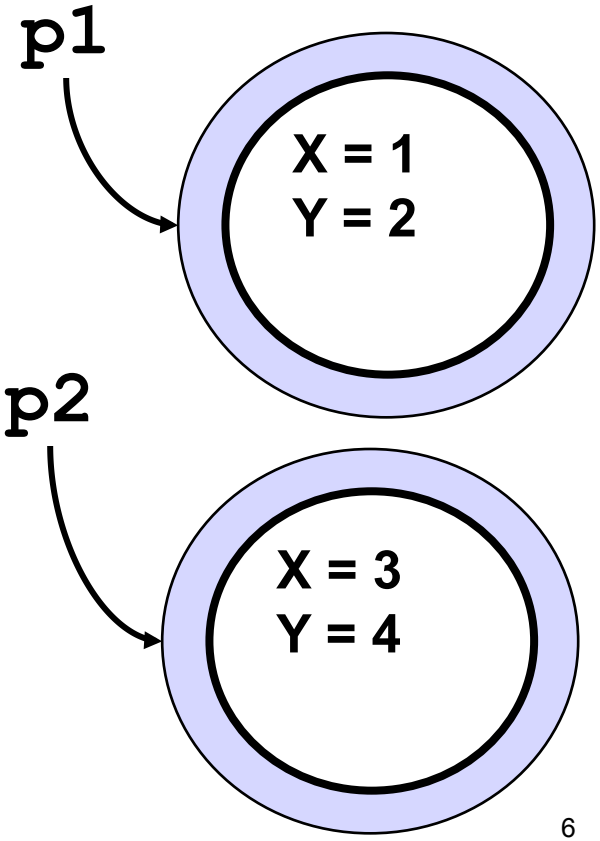
Nome do método deve ser igual ao nome da classe

Parâmetros

Uso de Construtores

- Deve ser usado no momento da criação do objeto:

```
Ponto p1 = new Ponto(1,2);

Ponto p2 = new Ponto(3,4);
```



Método Construtor Exemplo

```
class EditorGrafico
{

    public criarPontos ()
    {
        Ponto p1, p2;
        p1 = new Ponto (1,2);
        p2 = new Ponto (3,4);
    }
}
```

Construtor Padrão

- A linguagem Java declara um construtor padrão, vazio, que não recebe nenhum parâmetro

- Quando declaramos um novo construtor, esse construtor padrão deixa de existir e é substituído pelo novo construtor

Métodos Construtores

- Usados na criação de um objeto através do comando *new*
- **Possuem o mesmo nome da classe**
- Podem receber parâmetros que servirão para incialização dos atributos da classe
- Uma classe pode ter vários métodos construtores

Métodos Operacionais

- Implementam as funções de uma classe
- Possuem sintaxe semelhante à sintaxe de definição das funções de um programa procedural
- Determinam o comportamento da classe e a troca de mensagens com outras classes

Métodos Operacionais

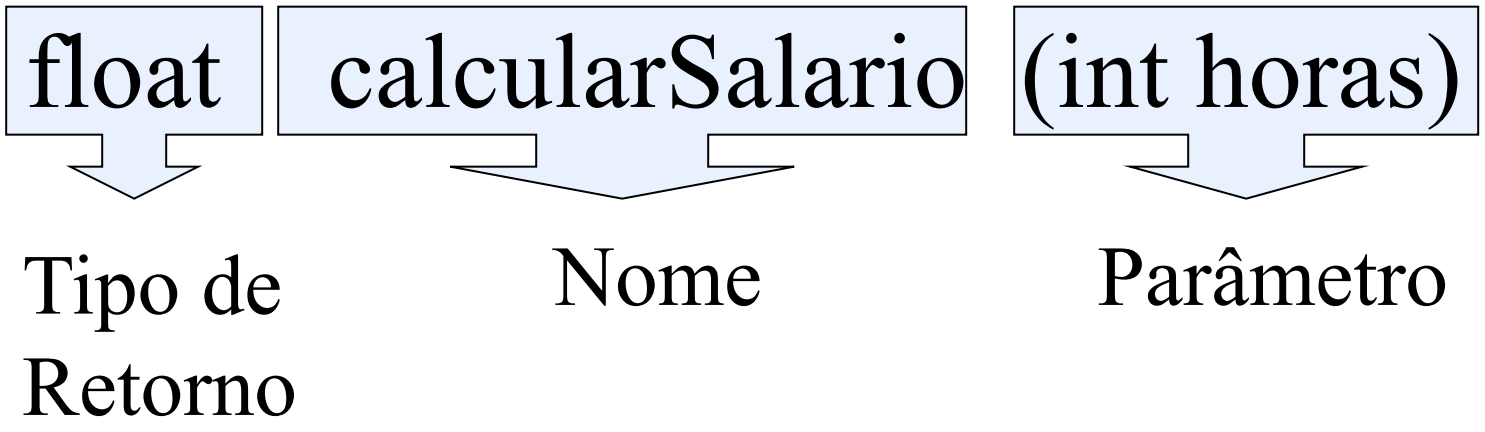
```
class Funcionario
{  String nomeFunc;
   float salario;
   ...
   float calcularSalario (int horas)
   {   float salMes = 0;
       if (horas < 220)
       {
        salMes =(salario/220)*horas;
       }
       return salMes;
   }
    ...
}
```

Assinatura

Corpo

Retorno

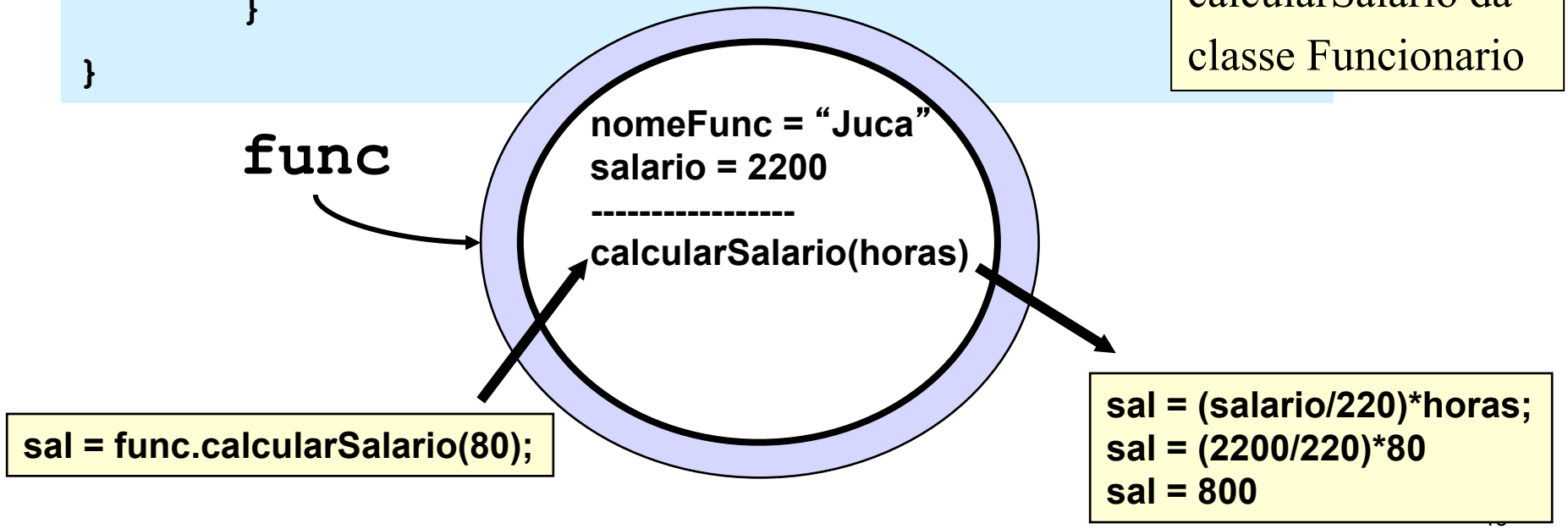
Assinatura de um método Operacional



Exemplo

```
class ProgramaPrincipal
{
        public static void main (String arg[])
        { float sal;
          Funcionario func;
          func = new Funcionario("Juca",2200);
          sal = func.calcularSalario(80);
          System.out.println (sal);
        }
}
```

Executando o Método calcularSalario da classe Funcionario



Sobrecarga de Métodos

- Sobrecarregar um método significa definir dois ou mais métodos com o mesmo nome, porém com assinaturas diferentes:
- Assinatura diferente pode ser: tipo de retorno, número ou tipos de parâmetros diferentes.

Sobrecarga de Métodos Construtores

- Exemplo:

```
Class Ponto {
    int x=0;
    int y=0;
    Ponto () {
    }
    Ponto (int x, int y) {
        this.x=x;
        this.y=y;
    }
}
```

Sobrecarga de Métodos Construtores

- A sobrecarga de construtores visa definir formas diferentes de criar um objeto
- Exemplo:
  - `Ponto p1 = new Ponto();`
  - `//p1 está em (0,0)`
  - `Ponto p2 = new Ponto(1,2);`
  - `//P2 está em (1,2)`

Encadeamento de Métodos Construtores

- Um construtor pode chamar outro construtor, isto se chama “encadeamento de métodos construtores”
- Para isto é necessário usar a seguinte construção: this(..). E esta chamada deverá ser a primeira linha do construtor.

Encadeamento de Métodos Construtores

- Exemplo:

```
Class Ponto {
    int x=0;
    int y=0;
    Ponto () {
      this(0,0);
    }
    Ponto (int x, int y) {
        this.x=x;
        this.y=y;
    }
}
```

Sobrecarga de Métodos Operacionais

- A sobrecarga pode ser feita igualmente aos métodos construtores

- Uma boa prática é usar a sobrecarga, somente, em métodos que possuam a mesma funcionalidade.

Sobrecarga de Métodos Operacionais

- Exemplo:

```
class Ponto {
        ...
      void mover (int dx, int dy){
          x += dx;
          y += dy;
      }
      void mover (int raio, float ang){
          x += raio*Math.cos(ang);
          y += raio*Math.sen(ang);
      }
}
```