

Assinatura de Métodos

Sobrecarga de Métodos

Métodos Construtores

Atributos e Métodos Estáticos



Assinatura

Identificação única
do método

Teste.java

```
class Teste {  
  
    int valor() {  
        return 2;  
    }  
  
    double valor() {  
        return 2.0;  
    }  
}
```

?

Meu programa

```
Teste t = new Teste();  
t.valor();
```

```
float meuMetodo() {  
  
}
```



meuMetodo()

```
float meuMetodo(int a, double b) {  
  
}
```



meuMetodo(int, double)

meuMetodo(1, 2.0)
int, double



~~meuMetodo(1.0, 2)~~
double, int



```
float meuMetodo(String a, int b, String c) {  
}
```



meuMetodo(String,int,String)

meuMetodo("oi", 5, "você")
String, int, String



meuMetodo("", 200, "")
String, int, String



~~meuMetodo("oi", "você", 4)~~
String, String, int





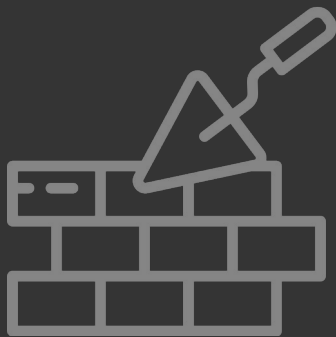
Sobrecarga

Métodos com mesmo nome, mas
diferentes assinaturas



Assinatura do método

```
class Teste {  
    int valor() { } ..... valor()  
        return 2;  
    }  
  
    double valor() { } ..... valor()  
        return 2.0;  
    }  
  
    double valor(int a) { } ..... valor(int)  
        return 2.0;  
    }  
  
    void valor(int a, double b) { } ..... valor(int, double)  
    }  
  
    void valor(double a, double b) { } ..... valor(double, double)  
    }  
}
```

Construtor

Método que “constrói” o objeto e o
deixa pronto para uso.

Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this.nome = new String("Alguém");  
    }  
}
```

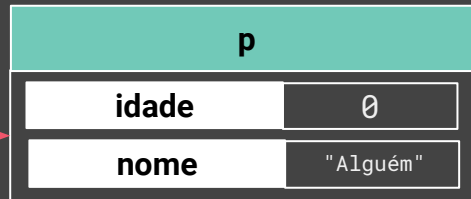
Mesmo nome da
classe e sem tipo
de retorno

Meu programa

```
Pessoa p = new Pessoa();  
System.out.println(p.nome);
```



Memória do meu programa



Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    // classe sem construtor  
}
```

compilador cria um
construtor padrão (sem
parâmetros)

Meu programa

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

```
public Pessoa() {  
  
}
```



Memória do meu programa

p	
idade	0
nome	null

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this.nome = new String("Alguém");  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Sobrecarga do
construtor

Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this.nome = new String("Alguém");  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Pessoa()

Pessoa(int)

Meu programa

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

Pessoa()

```
Pessoa t = new Pessoa(9);
```

Pessoa(int)



Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this.nome = new String("Alguém");  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Pessoa()



Pessoa(int)



Meu programa

```
Pessoa t = new Pessoa(3.0);
```

Pessoa(double) ❌

Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this.nome = new String("Alguém");  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Pessoa()

this()

Pessoa(int)

this(int)

Meu programa

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

Mesma linha
de código

Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this(0);  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Diagram illustrating the flow of calls between constructors:

- `Pessoa()` calls `this()` and `this(0)`.
- `this(0)` calls `Pessoa(int)`.
- `Pessoa(int)` calls `this(int)`.

Meu programa

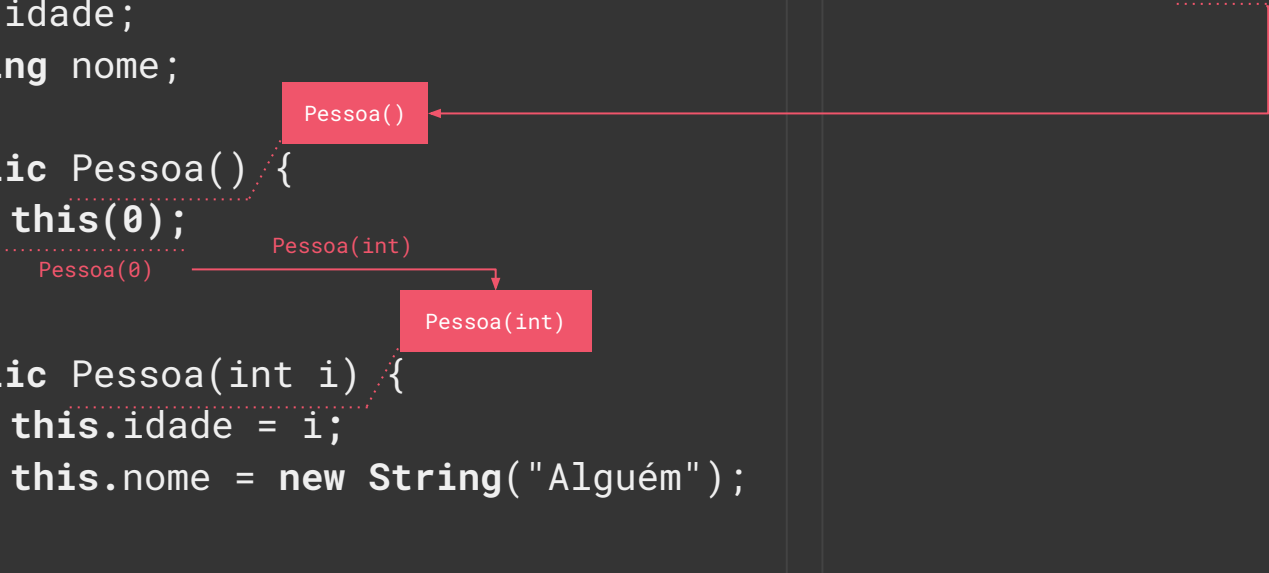
Um construtor
pode chamar
outro

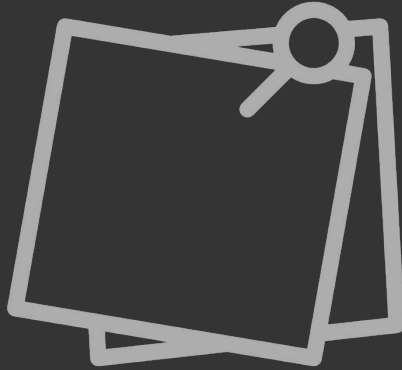
Pessoa.java

```
class Pessoa {  
    int idade;  
    String nome;  
  
    public Pessoa() {  
        this(0);  
    }  
  
    public Pessoa(int i) {  
        this.idade = i;  
        this.nome = new String("Alguém");  
    }  
}
```

Meu programa

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



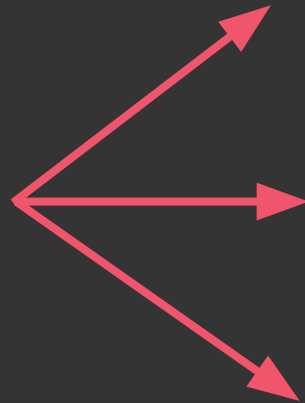


Atributos e métodos
estáticos



Classe

Membros estáticos:
Vinculados à classe (e não aos objetos
da classe)



objeto 1



objeto 2



objeto 3

```
class Pessoa {
```

```
    int idade;
```

```
    double peso;
```

Propriedades ou
atributos

```
    void oi() {
```

```
        System.out.println("oi");
```

```
    }
```

Método

```
}
```

Pessoa.java

```
class Pessoa {  
    int idade;  
    double peso;  
  
    void oi() {  
        System.out.println("oi");  
    }  
}
```

Meu programa

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

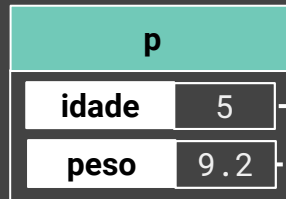
```
p.idade = 5; --
```

```
p.peso = 9.2; --
```

```
p.oi();
```



Memória do meu programa



Pessoa.java

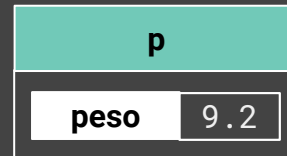
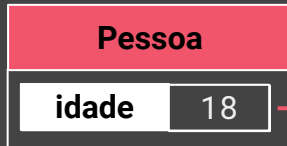
```
class Pessoa {  
    static int idade;  
    double peso;  
  
    void oi() {  
        System.out.println("oi");  
    }  
}
```

Meu programa

```
Pessoa p = new Pessoa();  
p.idade = 5;  
p.peso = 9.2;  
Pessoa.idade = 18;
```



Memória do meu programa



Pessoa.java

```
class Pessoa {  
    static int idade;  
    double peso;  
  
    static void oi() {  
        System.out.println("oi");  
    }  
}
```

Meu programa

Pessoa.oi();

Pessoa p = new Pessoa();
p.oi();



Memória do meu programa

Pessoa

idade

0

Matematica.java

```
class Matematica {  
    static int soma(int a, int b) {  
        int r = a + b;  
        return r;  
    }  
}
```

Teste.java

```
class Teste {  
    public static void main(String args[]) {  
        int x = Matematica.soma(2, 3);  
        System.out.println(x);  
    }  
}
```


Pessoa.java

```
class Pessoa {  
    static int idade;  
    double peso;  
  
    static void oi() {  
        System.out.println("oi");  
    }  
  
    void imprime() {  
        System.out.println(this.peso);  
        Pessoa.idade = 20;  
        Pessoa.oi();  
    }  
}
```

Teste.java

```
class Teste {  
    public static void main(String args[]) {  
        Pessoa p = new Pessoa();  
        p.imprime();  
    }  
}
```

Não pode usar this

Método estático não está vinculado ao espaço de memória de um objeto.

Pessoa

idade

20

p

peso

0.0

Matematica.java

```
class Matematica {  
    [ static double PI = 3.1415;  
}  
}
```

Teste.java

```
class Teste {  
    public static void main(String args[]) {  
        double raio = 3.0;  
        double area = Matematica.PI * raio;  
  
        System.out.println(area);  
    }  
}
```

Constantes

Propriedades estáticas são excelentes
para representar constantes

Teste.java

```
class Teste {  
    public static void main(String args[]) {
```



Algo familiar?

```
    }  
}
```

Créditos

Conteúdo

Fernando Bevilacqua
fernando.bevilacqua@uffs.edu.br

Ícones

DinosoftLabs
flaticon.com/authors/dinosoftlabs

mangsaabguru
flaticon.com/authors/mangsaabguru

Freepik
flaticon.com/authors/freepik

Smashicons
flaticon.com/authors/smashicons

Vitaly Gorbachev
flaticon.com/authors/vitaly-gorbachev