02

# Crie e sobrecarregue construtores

Construtores também podem ser sobrecarregados:

```
class Teste {
    public Teste() {
    }
    public Teste(int i) {
    }
}
```

COPIAR CÓDIGO

Cuidado com os exemplos de sobrecarga com `varargs`, como vimos antes, e no caso de herança.

Quando existem dois construtores na mesma classe, um construtor pode chamar o outro através da chamada `this`. Note que loops não compilam:

```
class Teste {
    public Teste() {
        System.out.println("construtor simples");
    }
    public Teste(int i) {
        this();
    }
    public Teste(String s) {
        this(s, s);  // não compila, loop
    }
```

```
    public Teste(String s, String s2) {
        this(s); // não compila, loop
    }
}
```

COPIAR CÓDIGO

Temos que tomar cuidado com sobrecarga da mesma maneira que tomamos cuidado com sobrecarga de métodos: os construtores invocados seguem as mesmas regras que as de métodos.

Quando um método utiliza `varargs`, se ele possui uma variação do método sem nenhum argumento e invocarmos sem argumento, ele chamará o método sem argumentos (para manter compatibilidade com versões anteriores do Java):

```
void desativa(Cliente... clientes) {
    System.out.println("varargs");
}
void desativa() {
    System.out.println("sem argumento");
}
void metodo() {
    desativa(); // imprime sem argumento
}
```

COPIAR CÓDIGO

A instrução `this` do construtor deve ser sempre a primeira dentro do construtor:

```
class Teste {
    Teste() {
        String valor = "valor...";
```

```
        this(valor); // não compila
    }

    Teste(String s) {
        System.out.println(s);
    }

    public static void main(String[] args) {
        new Teste();
    }
}
```

COPIAR CÓDIGO

Justo por isso não é possível ter duas chamadas a `this` :

```
class Teste {
    Teste() {
        this(valor);
        this(valor); // não compila
    }

    Teste(String s) {
        System.out.println(s);
    }

    public static void main(String[] args) {
        new Teste();
    }
}
```

COPIAR CÓDIGO

A instrução `this` pode envolver instruções:

```
class Teste {
    Teste() {
        this(valor());
    }

    private static String valor() {
        return "valor...";
    }

    Teste(String s) {
        System.out.println(s);
    }

    public static void main(String[] args) {
        new Teste();
    }
}
```

COPIAR CÓDIGO

A instrução não pode ser um método da própria classe, pois o objeto não foi construído ainda:

```
class Teste {
    Teste() {
        this(valor()); // valor não é estático, não compila
    }

    private String valor() {
        return "valor...";
    }

    Teste(String s) {
```

```
        System.out.println(s);
    }

    public static void main(String[] args) {
        new Teste();
    }
}
```

COPIAR CÓDIGO