

`this()` E `super()`

1. `this()`

- **O que é?** `this()` é uma palavra-chave usada em Java para chamar outro construtor da mesma classe. Ela é utilizada quando você quer reutilizar a lógica de um construtor dentro de outro, evitando a duplicação de código.
- **Como funciona?** Quando você chama `this()`, o Java invoca um outro construtor da mesma classe que corresponde à lista de parâmetros fornecida. Isso tem que ser feito na primeira linha do construtor.
- **Exemplo:**

```
class Dog {  
    Dog() {  
        System.out.println("Default Dog Constructor");  
    }  
  
    Dog(String name) {  
        this(); // Chama o construtor sem parâmetros da mesma classe  
        System.out.println("Dog's name is: " + name);  
    }  
}
```

Saída:

```
Default Dog Constructor  
Dog's name is: Rex
```

Aqui, `this()` chama o construtor padrão `Dog()`, que imprime "Default Dog Constructor", e depois continua com o código do segundo construtor.

2. `super()`

- **O que é?** `super()` é a palavra-chave usada em Java para chamar o construtor da classe pai (superclasse). Isso é essencial quando sua classe herda de outra, e você quer garantir que a parte da superclasse do objeto seja inicializada corretamente.
- **Como funciona?** Quando você chama `super()`, o Java invoca o construtor da superclasse que corresponde à lista de parâmetros fornecida. Também deve ser a primeira linha no construtor.
- **Exemplo:**

```
class Animal {  
    Animal() {  
        System.out.println("Animal Constructor");  
    }  
}  
  
class Dog extends Animal {  
    Dog() {  
        super(); // Chama o construtor da classe pai (Animal)  
        System.out.println("Dog Constructor");  
    }  
}
```

Saída:

```
Animal Constructor
Dog Constructor
```

Aqui, `super()` chama o construtor da classe `Animal`, que imprime "Animal Constructor", antes de executar o resto do código no construtor de `Dog`.

REGRAS DE USO `THIS()` E `SUPER()`

Regra: Se você não chamar explicitamente `this()` ou `super()` na primeira linha de um construtor, o compilador vai automaticamente inserir `super()`.

O que isso significa? Toda classe em Java herda da classe `Object` por padrão, exceto a classe `Object` em si. Então, se você não chamar `super()` ou `this()`, o compilador vai adicionar `super()` automaticamente, para garantir que o construtor da classe pai seja chamado.

Exemplo:

```
class Animal {
    Animal() {
        System.out.println("Animal Constructor");
    }
}

class Dog extends Animal {
    Dog() {
        // Aqui o compilador insere implicitamente super();
        System.out.println("Dog Constructor");
    }
}
```

Saída:

```
Animal Constructor
Dog Constructor
```

No exemplo acima, o compilador automaticamente insere `super()` ; na primeira linha do construtor `Dog()`, garantindo que o construtor da classe `Animal` seja chamado.

Nota:

Quando uma classe filha possui um construtor definido, o compilador Java não gera automaticamente um construtor padrão sem parâmetros na superclasse. Isso significa que, se a classe pai possui um construtor com parâmetros, você precisa chamar explicitamente o construtor da classe pai usando a instrução `super()` com os parâmetros necessários.

Exemplo:

```
class SuperClass {
    private int x;
    public SuperClass(int x) { this.x = x; }
}

class SubClass extends SuperClass {
```

```
public SubClass(int x) {
    super(x); // Chamada explícita ao construtor da superclasse
}
}
```

Nesse exemplo, como a classe `SuperClass` possui um construtor com parâmetros, o compilador não gera automaticamente um construtor padrão sem parâmetros. Portanto, na classe `SubClass`, é necessário chamar explicitamente o construtor da superclasse usando a instrução `super(x)`.

Regra: Você só pode chamar `super()` ou `this()` uma vez dentro de um construtor.

O que isso significa? Dentro de um construtor, ou você chama o construtor da classe pai (`super()`), ou você chama outro construtor da mesma classe (`this()`). Você não pode fazer as duas coisas ao mesmo tempo, e também não pode chamar `super()` ou `this()` mais de uma vez.

Exemplo Inválido:

```
class Dog extends Animal {
    Dog() {
        // Isso vai causar um erro de compilação
        super();
        this(); // Não pode chamar both super() e this()
    }
}
```

Exemplo Válido:

```
class Dog extends Animal {
    Dog() {
        super(); // Chama o construtor da classe pai
        System.out.println("Dog Constructor");
    }

    Dog(String name) {
        this(); // Chama o outro construtor da própria classe
        System.out.println("Dog Constructor with name: " + name);
    }
}
```

Nesse exemplo, no segundo construtor, `this()` chama o primeiro construtor `Dog()`. Você não pode ter `super()` e `this()` no mesmo construtor.

Regra: `super()` ou `this()` deve ser a primeira linha de qualquer construtor.

O que isso significa? Se você quiser chamar o construtor da classe pai ou outro construtor da mesma classe, isso deve ser feito logo no início, antes de qualquer outro código dentro do construtor.

Exemplo Inválido:

```
class Dog extends Animal {
    Dog() {
        System.out.println("Some message");
        super(); // Isso causa erro porque super() não está na primeira
linha
    }
}
```

Exemplo Válido:

```
class Dog extends Animal {
    Dog() {
        super(); // Chama primeiro o construtor da classe pai
        System.out.println("Dog Constructor");
    }
}
```

No exemplo válido, `super()` está na primeira linha, como deveria ser.

Vamos criar mais uns exemplos, para melhor entender a dinâmica dos construtores em Java:

CENÁRIO 1:

Neste exemplo eu chamo o construtor da classe pai **Parent** com parâmetro no construtor padrão da subclasse **Child**

```
1 package com.exemplo.thisandsuper.entidades;
2
3 public class Child extends Parent {
4
5     private int y;
6
7     public Child() {
8         super(10);
9         System.out.println("Construtor padrão da classe Child");
10    }
11
12    public Child(int y) {
13        this.y = y;
14        System.out.println("Construtor com parâmetro da classe Child: " + y);
15    }
16 }
17
18
19
20 package com.exemplo.thisandsuper.entidades;
21 public class Parent {
22     private int x;
23
24     public Parent() {
25         System.out.println("Construtor padrão da classe Parent");
26     }
27
28     public Parent(int x) {
29         this.x = x;
30         System.out.println("Construtor com parâmetro da classe Parent: " + x);
31     }
32 }
33
34
35 package com.exemplo.thisandsuper;
36
37 import com.exemplo.thisandsuper.entidades.Child;
38
39 public class MainClass {
40
41     public static void main(String[] args) {
42         Child child = new Child();
43     }
44 }
```

Saida Console:

```
<terminated> MainClass [Java Application] C:\Program Files\ eclipse-java-2022-09-R-win32.  
Construtor com parâmetro da classe Parent: 10  
Construtor padrão da classe Child
```

Veja que eu posso chamar qualquer construtor na classe filha, além do construtor padrão.

CENÁRIO 2:

Neste exemplo, não uso **super ()** no construtor padrão da subclasse **Child**, perceba que vai ser chamado **automaticamente** pelo compilador.

```
1 package com.exemplo.thisandsuper.entidades;  
2  
3 public class Child extends Parent {  
4  
5     private int y;  
6  
7     public Child() {  
8         System.out.println("Construtor padrão da classe Child");  
9     }  
10  
11     public Child(int y) {  
12         this.y = y;  
13         System.out.println("Construtor com parâmetro da classe Child: " + y);  
14     }  
15  
16 }
```

Saida Console:

```
Problems @ Javadoc Declaration Console X  
<terminated> MainClass [Java Application] C:\Program Files\ eclipse-java-2022-09-R-win32-x86_64\ eclipse\plugins\org.eclipse.justj.o  
Construtor padrão da classe Parent  
Construtor padrão da classe Child
```

CENÁRIO 3:

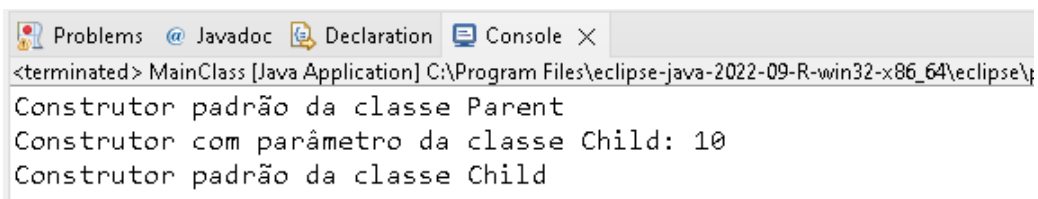
E se eu inserir `this(10)`, que chamará um construtor com parâmetro no construtor padrão da subclasse `Child`, o construtor padrão da superclasse `Parent` será chamado? Não posso inserir `this()` e `super()` pois ambos devem ser a 1ª linha do construtor. O que acontecerá?

```
1 package com.exemplo.thisandsuper.entidades;
2
3 public class Child extends Parent {
4
5     private int y;
6
7     public Child() {
8         this(10); // Chamada explícita ao construtor com parâmetro
9         System.out.println("Construtor padrão da classe Child");
10    }
11
12    public Child(int y) {
13        this.y = y;
14        System.out.println("Construtor com parâmetro da classe Child: " + y);
15    }
16
17 }
```

Quando você cria um objeto da classe `Child` usando o construtor padrão `new Child()`:

- O compilador procura o construtor padrão da classe `Child`.
- Nesse construtor, a primeira linha é a chamada a `this(10)`.
- Ao chamar `this(10)`, o compilador procura o construtor da classe `Child` que aceita um parâmetro do tipo `int`.
- Nesse construtor, não há nenhuma chamada a `super()`, então o compilador irá inserir uma chamada implícita a `super()`.
- Então, o construtor padrão da classe `Parent` é chamado primeiro, imprimindo "Construtor padrão da classe Parent".
- Em seguida, o construtor com parâmetro da classe `Child` é executado, imprimindo "Construtor com parâmetro da classe Child: 10".
- Por fim, o construtor padrão da classe `Child` é executado, imprimindo "Construtor padrão da classe Child".

Então, mesmo sem a chamada explícita a `super()` no construtor com parâmetro da classe `Child`, o compilador irá inserir uma chamada implícita a `super()`, garantindo que o construtor da classe pai seja chamado.



```
<terminated> MainClass [Java Application] C:\Program Files\ eclipse-java-2022-09-R-win32-x86_64\ eclipse\
Construtor padrão da classe Parent
Construtor com parâmetro da classe Child: 10
Construtor padrão da classe Child
```

Podemos também explicitamente chamar `super()` no construtor com parâmetros que o efeito será o mesmo, ou mesmo chamar um construtor da superclasse com parâmetros

(ex: `super(50)`) e o resultado será o mesmo, com exceção de que o construtor chamado da superclasse não será o construtor padrão.

```
1 package com.exemplo.thisandsuper.entidades;
2
3 public class Child extends Parent {
4
5     private int y;
6
7     public Child() {
8         this(10); // Chamada explícita ao construtor com parâmetro
9         System.out.println("Construtor padrão da classe Child");
10    }
11
12    public Child(int y) {
13        super(50);
14        this.y = y;
15        System.out.println("Construtor com parâmetro da classe Child: " + y);
16    }
17
18 }
```

Problems Javadoc Declaration Console X

<terminated> MainClass [Java Application] C:\Program Files\ eclipse-java-2022-09-R-win32-x86_64\ eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32

Construtor com parâmetro da classe Parent: 50
Construtor com parâmetro da classe Child: 10
Construtor padrão da classe Child