

this() E super()

1. this()

- `this()` é uma palavra-chave que pode ser 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.
- 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);
    }
}

public class Exemplo {
    public static void main(String[] args) {
        Dog dog = new Dog("Rex");
    }
}
```

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()

- `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.
- 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");
    }
}
```

```

    }

    public class Exemplo {
        public static void main(String[] args) {
            Dog dog = new Dog();
        }
    }

```

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`. A declaração explícita do construtor *default* da classe pai não é obrigatória neste caso, declaramos de forma explícita apenas por causa do `System.out.println()`

REGRAS DE USO THIS () E SUPER ()

Em qualquer construtor onde é necessária a chamada de outro construtor, seja de uma classe base ou derivada, a primeira instrução deve ser uma chamada a outro construtor (da mesma classe com `this()` ou da classe pai com `super()`).

Se você não fizer isso explicitamente, o compilador inserirá uma chamada implícita para `super()`, que invoca o construtor sem argumentos da classe pai.

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");
    }
}

public class Exemplo {
    public static void main(String[] args)
        Dog dog = new Dog();
    }
}

```

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.

IMPORTANTE:

1. Geração do construtor padrão: Quando você define explicitamente qualquer construtor em uma classe (seja ela uma classe pai ou filha), o Java **não gera mais o construtor padrão** sem parâmetros automaticamente para essa classe.
2. Herança e construtores: Quando uma classe filha herda de uma classe pai, é importante entender como os construtores se comportam:
 - Se a classe pai possui apenas construtores com parâmetros, a classe filha deve chamar explicitamente um desses construtores usando a instrução `super()` com os parâmetros necessários.
 - O compilador Java não gera automaticamente uma chamada para um construtor sem parâmetros da superclasse se este não existir. Portanto se uma classe pai possui apenas construtores com parâmetros e a classe filha tenta chamar `super()` sem argumentos (assumindo erroneamente que existe um construtor padrão), **ocorrerá um erro de compilação**.

Exemplo A:

```

3 class Animal {
4
5     Animal(String nome)
6     {
7         System.out.println(nome);
8     }
9 }
10
11 class Dog extends Animal {
12     Dog() {
13         super();
14     }
15 }
16
17 // The constructor Animal() is undefined
18 // 3 quick fixes available:
19 // + Add argument to match 'Animal(String)'
20 // - Change constructor 'Animal(String)': Remove parameter 'String'
21 // + Create constructor 'Animal()'
22
23 public class Main {
24     public static void main(String[] args) {
25         // Criando uma instância da classe Dog, o construtor do Animal será chamado primeiro
26         Dog dog = new Dog();
27     }
28 }

```

O código acima é um exemplo que demonstra perfeitamente que:

- Quando uma superclasse tem apenas construtores com parâmetros, as subclasses devem chamar explicitamente um desses construtores (veja próximo exemplo abaixo).
- Não se pode assumir que existe um construtor padrão na superclasse.
- Tentar usar `super()` sem argumentos quando não há um construtor padrão na superclasse resulta em um erro de compilação.

Exemplo B:

Da mesma forma, ao gerar um construtor com parâmetro para a classe filha, o construtor *default* deixa de existir e a chamada de `this()` dentro do construtor com parâmetro gera erro de compilação:

```

3 class Animal {
4     Animal(){
5         System.out.println("Animal Constructor");
6     }
7 }
8
9 }
10
11 class Dog extends Animal {
12
13     Dog(String nome) {
14         this();
15         System.out.println("Dog Constructor: " + nome);
16     }
17 }
18
19
20 public class Exemplo {
21     public static void main(String[] args) {
22
23         Dog dog = new Dog("Toto");
24     }
25 }
26
27

```

Exemplo C:

Aqui neste exemplo chamamos o construtor com parâmetro explicitamente:

```

class Animal {
    Animal(String nome) {
        System.out.println("Animal Constructor com nome: " + nome);
    }
}

class Dog extends Animal {
    Dog() {

        super("Rex"); // Adicionando uma chamada explícita com um argumento
        System.out.println("Dog Constructor");
    }
}

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

        Dog dog = new Dog();
    }
}

```

Saída:

```

Animal Constructor com nome: Rex
Dog Constructor

```

Exemplo D:

Neste caso o `super()` não é nem necessário, é chamado automaticamente pelo Java pois o construtor *default* é declarado explicitamente:

```

class Animal {
    Animal(String nome) {
        System.out.println("Animal Constructor com nome: " + nome);
    }
}

```

```

// declaração explícita do construtor default é necessária
// pois existe outro construtor com parâmetros
Animal( ) {
    System.out.println("Animal Constructor default");
}

class Dog extends Animal {
    Dog() {

        // se nenhum construtor da superclasse é chamado, o construtor
        // default é chamado automaticamente
        System.out.println("Dog Constructor");
    }
}

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

        Dog dog = new Dog();
    }
}

```

Saída:

```

Animal Constructor default
Dog Constructor

```

Lembre-se: Ao instanciar uma classe em Java, **um construtor é sempre chamado**, seja ele o construtor padrão (*default*) ou um construtor com parâmetros, garantindo que a inicialização do objeto ocorra.

Se não houver um construtor definido explicitamente, o compilador fornecerá um construtor padrão sem parâmetros. Se você definir qualquer construtor com parâmetros, o compilador não criará o construtor *default* implicitamente, e você precisará definir um se necessário.

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

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:

```

3  class Animal {
4      Animal(){
5          System.out.println("Animal Constructor");
6      }
7  }
8
9  }
10
11 class Dog extends Animal {
12     Dog() {
13         System.out.println("Dog Constructor");
14     }
15
16     Dog(String name) {
17         this(); // Chama o outro construtor da própria classe
18         super();
19     }
20 }
21
22
23
24
25 public class Exemplo {
26     public static void main(String[] args) {
27
28         Dog dog = new Dog("Toto");
29     }
30 }
31

```

Constructor call must be the first statement in a constructor with name: " + name);

Exemplo Válido:

```

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

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);
    }
}

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

        Dog dog = new Dog("Toto");
    }
}

```

Saída:

```

Animal Constructor
Dog Constructor
Dog Constructor with name: Toto

```

Nesse exemplo, no segundo construtor, `this()` chama o primeiro construtor `Dog()`, que por sua vez chama o construtor da superclasse. Você não pode ter `super()` e `this()` no mesmo construtor.

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

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 }
```

```

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

```

```

1 package com.exemplo.thisandsuper;
2
3 import com.exemplo.thisandsuper.entidades.Child;
4
5 public class MainClass {
6
7     public static void main(String[] args) {
8         Child child = new Child();
9     }
10 }

```

Saída 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 da superclasse , além do construtor padrão.

CENÁRIO 2:

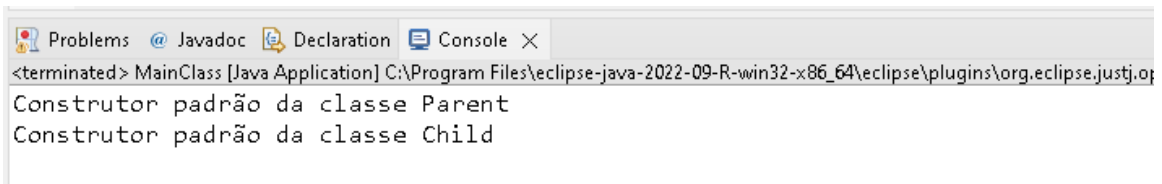
Neste exemplo, não uso **super ()** no construtor padrão da subclasse **Child**, perceba que vai ser chamado **automaticamente** pelo compilador.

```

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 }

```

Saída 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 }

```

Saída Console:

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.

```

Problems @ Javadoc Declaration Console X
<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.win3
Construtor com parâmetro da classe Parent: 50
Construtor com parâmetro da classe Child: 10
Construtor padrão da classe Child

```