02

Diferença entre o construtor padrão e construtores definidos pelo usuário

Quando não escrevemos um construtor na nossa classe, o compilador nos dá um construtor padrão. Esse construtor, chamado de *default* não recebe argumentos, tem a mesma visibilidade da classe e tem a chamada a `super()` .

A classe a seguir:

```
class A {
}
```

COPIAR CÓDIGO

... na verdade, acaba sendo:

```
class A {
    A() {
        super();
    }
}
```

COPIAR CÓDIGO

Caso você adicione um construtor qualquer, o construtor `default` deixa de existir:

```
class A {}
class B {
    B(String s) {}
}
class Teste {
    public static void main(String[] args) {
        new A(); // construtor padrão, compila
        new B(); // não existe mais construtor padrão
        new B("CDC"); // construtor existente
    }
}
```

COPIAR CÓDIGO

Dentro de um construtor você pode acessar e atribuir valores aos atributos, suas variáveis membro:

```
class Teste {
    int i;
    Teste() {
        i = 15; // agora i vale 15
        System.out.println(i); // 15
    }

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

COPIAR CÓDIGO

Os valores inicializados com a declaração das variáveis são inicializados **antes** d construtor, justamente por isso o valor inicial de `i` é 0, o valor padrão de uma

variável `int` membro:

```
class Teste {
    int i;
    Teste() {
        System.out.println(i); // vale 0 por padrão
        i = 15; // agora i vale 15
        System.out.println(i); // 15
    }

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

COPIAR CÓDIGO

Vale lembrar que variáveis membro são inicializadas automaticamente para: numéricas 0, `boolean` `false`, referências `null`.

Cuidado ao acessar métodos cujas variáveis ainda não foram inicializadas no construtor. O exemplo a seguir mostra um caso em que o método de inicialização é invocado antes de setar o valor da variável no construtor, o que causa um `NullPointerException`.

```
class A {

    int i = 15;
    String nome;
    int tamanho = tamanhoDoNome();

    A(String nome) {
```

```
        this.nome = nome;
    }

    int tamanhoDoNome() {
        return nome.length();
    }

    A() {
    }

}
```

COPIAR CÓDIGO

Mesmo que inicializemos a variável fora do construtor, após a chamada do método pode ocorrer um erro, como no caso a seguir, de um outro `NullPointerException`:

```
class A {

    int i = 15;
    String nome;
    int tamanho = tamanhoDoSobrenome();
    String sobrenome = "Silveira";

    A(String nome) {
        this.nome = nome;
    }

    int tamanhoDoSobrenome() {
        return sobrenome.length();
    }

    A() {
```

```
    }

}
```

COPIAR CÓDIGO

Mudar a ordem da declaração das variáveis resolve o problema, uma vez que o método é agora invocado após a inicialização da variável `sobrenome` :

```
class A {

    int i = 15;
    String nome;
    String sobrenome = "Silveira";
    int tamanho = tamanhoDoSobrenome();

    A(String nome) {
        this.nome = nome;
    }

    int tamanhoDoSobrenome() {
        return sobrenome.length();
    }

    A() {
    }

}
```

COPIAR CÓDIGO

Cuidado ao invocar métodos no construtor e variáveis estarem nulas:

```
class Teste {
    String nome;
    Teste() {
        testaTamanho(); // NullPointerException
        nome = "aprendendo";
    }

    private void testaTamanho() {
        System.out.println(nome.length());
    }

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

COPIAR CÓDIGO

E mais cuidado ainda caso isso ocorra por causa de sobrescrita de método, em que também poderemos ter essa `Exception`:

```
class Base {
    String nome;
    Base() {
        testa();
        nome = "aprendendo";
    }

    void testa() {
        System.out.println("testa");
    }

}
```

```
class Teste extends Base {
    void testa() {
        System.out.println(nome.length());
    }
    public static void main(String[] args) {
        new Teste();
    }
}
```

COPIAR CÓDIGO

Já se o método `testa` for privado, como o *binding* da chamada ao método é feito em compilação, o método invocado pelo construtor é o da classe mãe, sem dar a `Exception`:

```
class Base {
    String nome;
    Base() {
        testa();
        nome = "aprendendo";
    }

    private void testa() {
        System.out.println("testa");
    }

}
class Teste extends Base {
    void testa() {
        System.out.println(nome.length());
    }
    public static void main(String[] args) {
        new Teste();
```

```
    }
}
```

COPIAR CÓDIGO

Você pode entrar em loop infinito, cuidado, `StackOverflow` :

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

COPIAR CÓDIGO

Construtores podem ser de todos os tipos de modificadores de acesso: `private` , `protected` , `default` e `public` .

É comum criar um construtor privado e um método estático para criar seu objeto:

```
class Teste {
    private Teste() {
    }

    public static Teste cria() {
        return new Teste();
    }
}
```

COPIAR CÓDIGO

Tenha muito cuidado com um método com nome do construtor. Se colocar um `void` na frente, vira um método:

```
class Teste {
  void Teste() {
    System.out.println("Construindo");
  }

  public static void main(String[] args) {

    new Teste();
    // não imprime nada, definimos um método e não o
construtor
    new Teste().Teste();
    // agora imprime Construindo
  }
}
```

COPIAR CÓDIGO

Existem também blocos de inicialização que não são cobrados na prova.