03

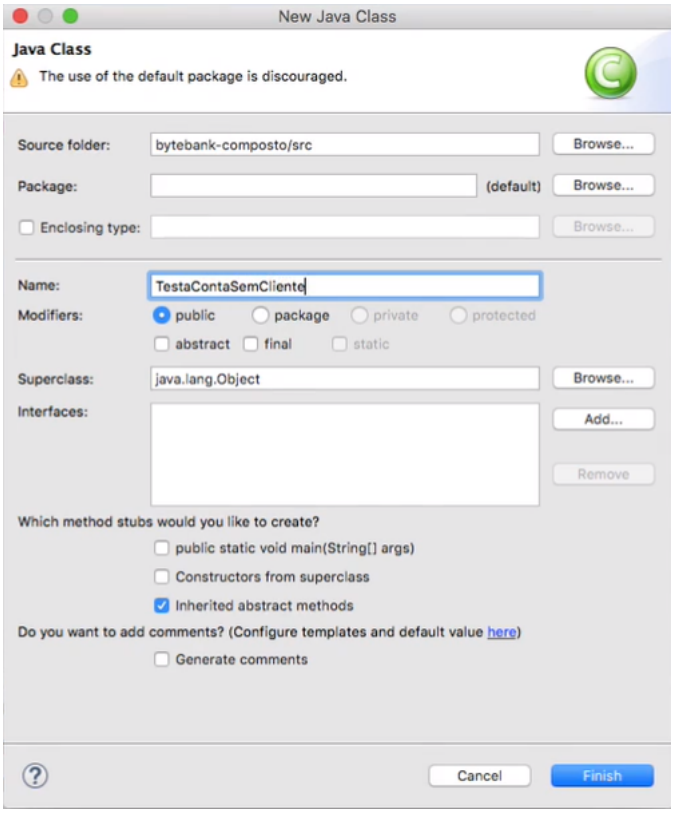
Referência Null

Transcrição

Voltaremos para a questão do valor `0` dos atributos e como isso se dá nos casos do tipo `String`. Qual seria o valor zerado do atributo `titular` da classe `Conta`, uma vez que ela faz referência à outra classe `Cliente`?

```
public class Conta {
    double saldo;
    int agencia;
    int numero;
    Cliente titular;

    // ...
```

Realizaremos um teste para tentar descobrir o que acontece caso não criemos um `Cliente`. Para isso, criaremos uma nova classe chamada `TestaContaSemCliente`.



Criaremos na nova classe o `main`, bem como uma referência para `Conta` chamada `contaDaMarcela` utilizando a palavra-chave `new`.

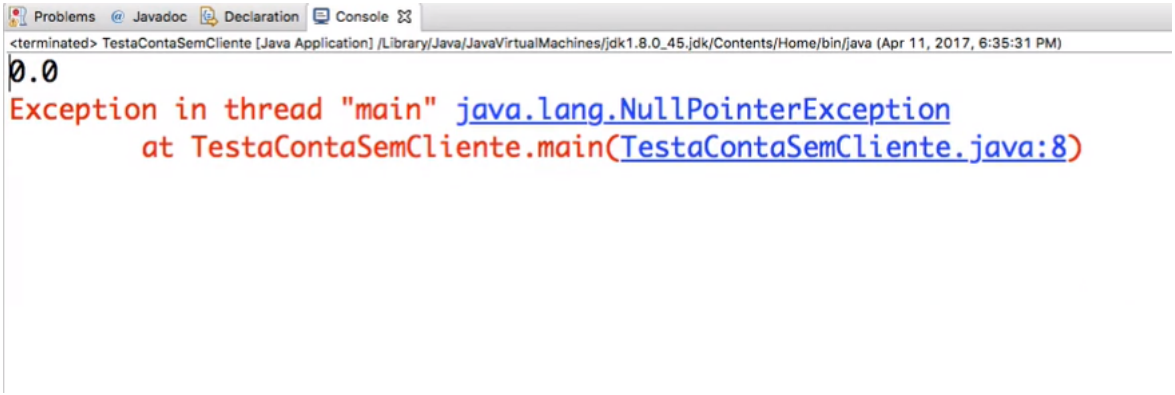
```
public class TestaContaSemCliente {
    public static void main(String[] args) {
        Conta contaDaMarcela = new Conta();
        System.out.println(contaDaMarcela.saldo);
    }
}
```

Acionando o `sysout` para `saldo`, o código é compilado e o atributo zerado sem nenhum problema. Mas se tentarmos fazer um procedimento parecido com `titular`, sem definirmos um `Cliente` para este atributo?

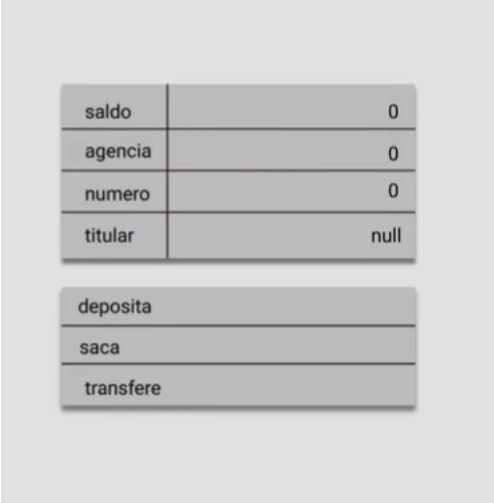
```
public class TestaContaSemCliente {
    public static void main(String[] args) {
        Conta contaDaMarcela = new Conta();
        System.out.println(contaDaMarcela.saldo);

        contaDaMarcela.titular.nome = "Marcela";
        System.out.println(contaDaMarcela.titular.nome);
    }
}
```

Ao executarmos a aplicação, veremos que houve uma mensagem de erro. Por enquanto, o conteúdo dessa mensagem ficará nebuloso, mas analisaremos os termos da mensagem posteriormente.

Percebam que está em destaque a linha que ocasionou o erro de aplicação, que é `TestaContaSemClinete.java:8`.



O "zero" de um atributo ou variável do tipo referência, chamamos de `null`, que significa algo como "referência para lugar nenhum".



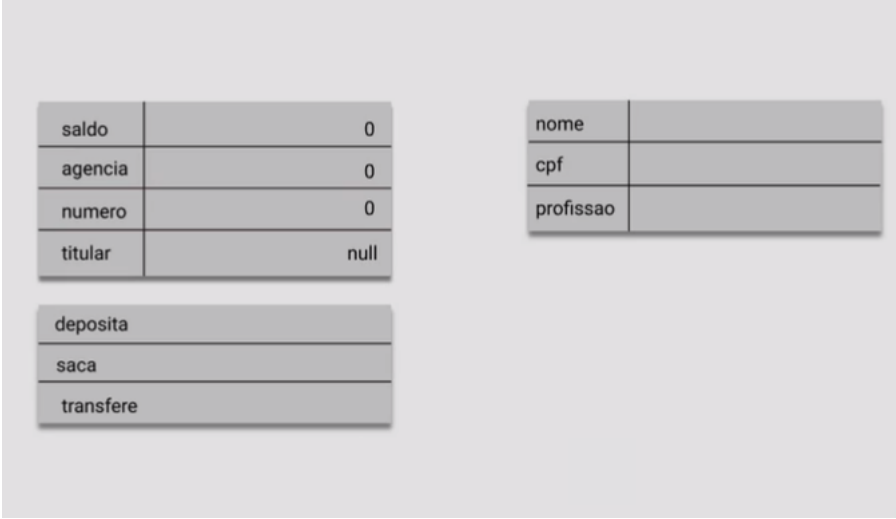
Podemos ter uma referência para nada no nosso código. Na linha 7 do nosso código, podemos realizar um `sysout` em `contaDaMarcela` e `titular`.

```
public class TestaContaSemCliente {
    public static void main(String[] args) {
        Conta contaDaMarcela = new Conta();
```

```
        System.out.println(contaDaMarcela.saldo);

        System.out.println(contaDaMarcela.titular);

        contaDaMarcela.titular.nome = "Marcela";
        System.out.println(contaDaMarcela.titular.nome);
    }
}
```

Ao executarmos novamente o código, veremos que o resultado da impressão será a mensagem `null` antes do surgimento do erro.

O nome do `titular` (`"Marcela"`) não faz referência a nenhum tipo `Cliente` . Para que a aplicação seja executada corretamente, precisamos criar um novo cliente e fazer a associação entre `Conta` e `Cliente` .

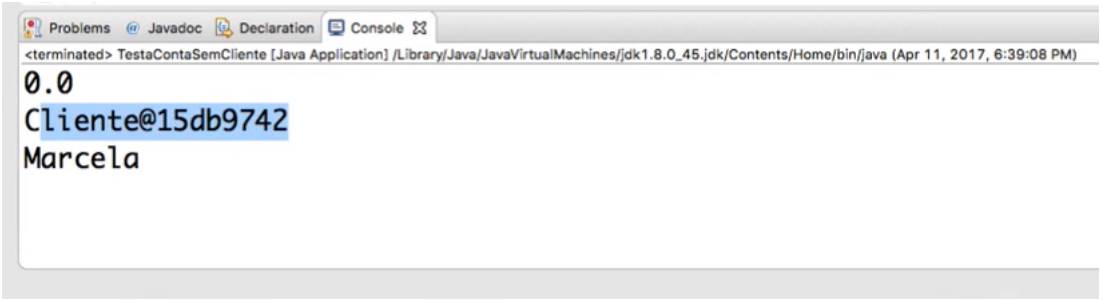


Já temos no código a associação ao objeto `Conta` através da variável `contaDaMarcela` . Nosso próximo passo é fazer a associação entre `titular` e `nome` . Nas atividades anteriores, declaramos que a variável `paulo` era responsável por essa associação entre objetos. Neste caso, faremos de outro modo.

Faremos com que `titular` deixe de ser `null`, fazendo-o receber um novo cliente: `new Cliente`.

```
public class TestaContaSemCliente {
    public static void main(String[] args) {
        Conta contaDaMarcela = new Conta();
        System.out.println(contaDaMarcela.saldo);

        contaDaMarcela.titular = new Cliente();
        System.out.println(contaDaMarcela.titular);

        contaDaMarcela.titular.nome = "Marcela";
        System.out.println(contaDaMarcela.titular.nome);
    }
}
```

Há casos em que não há necessidade de criar uma variável temporária, podemos criar a associação em uma linha, como é o caso. Ao executarmos a aplicação, será impresso um `Id` referente ao `Cliente`, revelando a associação feita entre os objetos.



Normalmente no Java, são criadas grandes redes de objetos interconectados que se referenciam, e através da invocação de métodos conseguimos fazer com que eles trabalhem entre si. O resultado é que temos códigos curtos, mas que atuam em grandes conjuntos. Com isso, temos uma maior organização no projeto, pois são códigos mais fáceis de ler e de realizar manutenções.

O `null` é uma referência que você encontrará com muita frequência, e não há necessidade de se preocupar com ela.

Uma referência é tida como `null` porque ainda não foi populada.

Para popular uma referência basta inserirmos um valor dentro dela, normalmente através de um `new` ou apontando para uma referência já existente, como fizemos no código anterior com `paulo`.

Um último desafio: lembre-se que podemos setar valores *default*. Na classe `Conta`, podemos dizer que toda vez que uma conta é aberta no ByteBank o saldo se inicia com `100`.

```
public class Conta {
    double saldo = 100;
    int agencia;
    int numero;
    Cliente titular;

    // ...
```

Do mesmo modo, podemos fazer com o que toda a vez que o `new` é acionado para uma `Conta`, temos um novo `Cliente`. Ou seja, toda `Conta` já se associa a um `Cliente`, com isso, não nos preocuparíamos com o `null` neste caso em particular.

```
public class Conta {
    double saldo = 100;
    int agencia;
    int numero;
    Cliente titular = new Cliente();

    // ...
```

No nosso projeto não é uma opção muito interessante, pois toda a conta tem de ser associada à um cliente novo, banindo a possibilidade de um cliente ter duas contas, por exemplo. Porém, em muitos casos, essa é uma alternativa interessante.