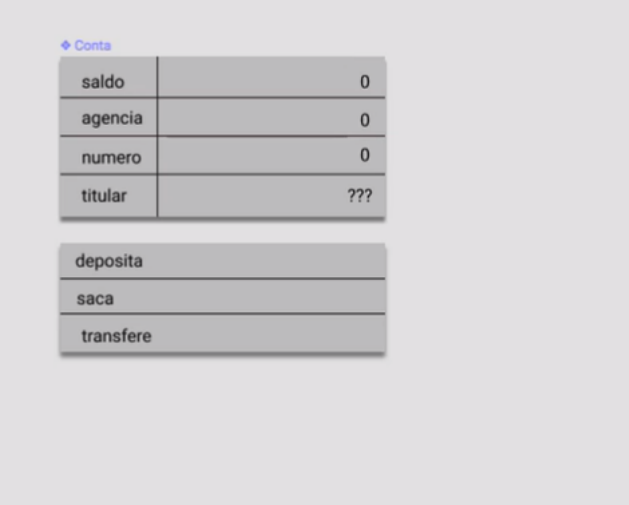
08

Métodos com referência e mais retorno

Transcrição

Tenho um desafio para que você saiba se entendeu o assunto **métodos**.

De acordo com o diagrama de contas, chegamos ao nosso terceiro método: o `transfere()`, que tem como finalidade **transferir dinheiro** de uma conta para outra.



Vamos incluir o novo método na classe `Conta`.

Abaixo do método `saca()`, adicionaremos `transfere()`.

Pensaremos sobre um ponto: quando transferimos dinheiro de uma conta para outra, receberemos uma informação de volta? Neste caso, sim. Para que uma transferência seja realizada com sucesso, é necessário haver dinheiro o suficiente em uma conta. Caso contrário, o retorno será `false`.

Usaremos o tipo `boolean`, com uma finalidade parecida com a do método `saca()`.

```
public boolean saca(double valor){
    if(this.saldo >= valor) {
        this.saldo -= valor;
        return true;
    } else {
        return false
    }
}

public boolean transfere() {

}
```

E quais argumentos o método recém-criado receberá? Nós, certamente, transferiremos um valor, expressaremos isso adicionando a variável `valor` que será do tipo `double`.

```
public boolean transfere(double valor, Conta destino) {

}
```

Observe que adicionamos um segundo argumento, que foi separado por uma vírgula (`,`) do primeiro. Foi necessário incluir um segundo argumento no método referente a **conta de destino** do depósito, ou seja, a conta para onde o dinheiro será transferido.

No entanto, será que é preciso especificar a conta de onde o dinheiro será retirado? Quantos argumentos serão necessários?

Nós adicionamos a `Conta destino` . Pela primeira vez, recebemos uma variável que usa também letras maiúsculas. Falta nela a tipagem de `double` ou `int` como fizemos anteriormente. Agora, estamos fazendo uma referência para outra conta.

Será que devemos receber a `Conta origem` ? Não. E por quê?

Antes, você vai perceber que a linha com `transfere()` está vermelha. Isto ocorre, porque nosso código não está sendo compilado, afinal, falta adicionar o retorno (`return`).

Em seguida, acessaremos `TestaMetodo.java` . Abaixo do último `println` de `conseguiuRetirar` , criaremos uma referência a uma outra conta, que chamaremos de `contaDaMarcela` .

Será nesta conta que depositaremos `1000` reais. Nosso objetivo será transferir `300` reais de `contaDaMarcela` para `contaDoPaulo` .

O interessante da orientação a objetos é que a nossa primeira frase será uma *referência* para a variável.

```
public class TestaMetodo {
    public static void main(String[] args) {
        Conta contaDoPaulo = new Conta();
        contaDoPaulo.saldo = 100;
        contaDoPaulo.deposita(50);
        System.out.println(contaDoPaulo.saldo);

        boolean conseguiuRetirar = contaDoPaulo.saca(20);
        System.out.println(contaDoPaulo.saldo);
        System.out.println(conseguiuRetirar);

        Conta contaDaMarcela = new Conta();
        contaDaMarcela.deposita(1000);
    }
}
```

Primeiramente, escreveremos `contaDaMarcela` e depois, passaremos o método `transfere()` .

Enquanto digitamos o código, o *autocomplete* do Eclipse vai disponibilizar diversos métodos, incluindo `transfere()` , que ainda tem um erro no outro arquivo.

No entanto, se tentássemos já executar o código, receberíamos um aviso de possível problema na compilação.

Como parâmetro, passaremos o valor de `300`, que será transferido, e como parâmetro de destino, incluiremos uma referência de `contaDoPaulo`. Foi desnecessário colocar, por exemplo, o `id` da conta do Paulo. O método vai receber um variável do tipo conta, que não é um objeto `Conta`.

Você pode ter ficado com a impressão de que estamos passando uma conta dentro de um método. Este não é o caso. Nós mandamos um "número interno" que o Java enxerga e não precisa tanto da nossa atenção.

```
public class TestaMetodo {
    public static void main(String[] args) {
        Conta contaDoPaulo = new Conta();
        contaDoPaulo.saldo = 100;
        contaDoPaulo.deposita(50);
        System.out.println(contaDoPaulo.saldo);

        boolean conseguiuRetirar = contaDoPaulo.saca(20);
        System.out.println(contaDoPaulo.saldo);
        System.out.println(conseguiuRetirar);

        Conta contaDaMarcela = new Conta();
        contaDaMarcela.deposita(1000);

        contaDaMarcela.transfere(300, contaDoPaulo);
    }
}
```

Evitamos adicionar `ContaDaMarcela`, porque a conta de origem é o valor no primeiro parâmetro. Ou seja, vamos passar apenas a conta de destino, considerando que a origem já é o objeto ao qual invocamos o método.

Seria equivalente a referência `this` do método `transfere()` no outro lado. Agora, só falta implementá-lo na classe `Conta`.

Para isto, usaremos `if`: caso o `saldo` de `this` - fazendo referência a `contaDaMarcela` - seja igual ou superior ao `valor` de transferência, será subtraído de `saldo` o `valor` referente à transferência.

```
public class Conta {

    // atributos
    // métodos

    public boolean transfere(double valor, Conta destino) {
        if(this.saldo >= valor) {
            this.saldo -= valor;
        }

    }
}
```

Por enquanto, apenas retiramos um `valor` de `saldo`, precisaremos que este valor seja transferido para uma conta destino, no caso, `contaDoPaulo`. Existem duas formas de executar essa função, uma delas é esta:

```
destino.saldo += valor;
```

Ou, podemos reutilizar um método da classe `Conta`, o `deposita()`.

Retiramos um `valor` e ele foi depositado. Neste caso, a transferência foi realizada com sucesso, e o retorno será `true`. Caso não haja dinheiro o suficiente para realizar a transferência, será retornado `false`.

A utilização do `else` é opcional no código, o retorno `false` ocorrerá mesmo que esta palavra não tenha sido utilizada.

```
public class Conta {

    // atributos
    // métodos

    public boolean transfere(double valor, Conta destino) {
        if(this.saldo >= valor) {
            this.saldo -= valor;
            destino.deposita(valor);
            return true;
        } else {
            return false;
        }
    }
}
```

Em `TestaMetodo`, acionaremos o `sysout` para `contaDaMarcela`.

```
public class TestaMetodo {
    public static void main(String[] args) {
        Conta contaDoPaulo = new Conta();
        contaDoPaulo.saldo = 100;
        contaDoPaulo.deposita(50);
        System.out.println(contaDoPaulo.saldo);

        boolean conseguiuRetirar = contaDoPaulo.saca(20);
        System.out.println(contaDoPaulo.saldo);
        System.out.println(conseguiuRetirar);

        Conta contaDaMarcela = new Conta();
        contaDaMarcela.deposita(1000);

        contaDaMarcela.transfere(300, contaDoPaulo);
        System.out.println(contaDaMarcela.saldo);
    }
}
```

Ao executarmos o programa veremos que o valor impresso será `700`, ou seja, nosso código funcionou perfeitamente.

Podemos, inclusive, verificar se no `saldo` de `contaDoPaulo` há `300` reais a mais, como o esperado. Para isso. acionaremos o `sysout` novamente para `contaDoPaulo`.

```
    contaDaMarcela.transfere(300, contaDoPaulo);
    System.out.println(contaDaMarcela.saldo);
```

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

Veremos que o valor impresso ao final da execução será de `430` reais, comprovando o sucesso do método. Não utilizamos o valor devolvido pelo *boolean*, poderíamos ter guardado esse valor através da utilização de `if`. Estamos invocando o método `transfere()` - lembre-se que dentro de `if` só são possíveis expressões booleanas - que devolve *boolean*, e portanto, o `if` pode ser compilado.

Caso o resultado dessa transferência tenha dado `true`, iremos imprimir "transferência com sucesso". Caso contrário, imprimiremos "faltou dinheiro".

```
public class TestaMetodo {
    public static void main(String[] args) {
        Conta contaDoPaulo = new Conta();
        contaDoPaulo.saldo = 100;
        contaDoPaulo.deposita(50);
        System.out.println(contaDoPaulo.saldo);

        boolean conseguiuRetirar = contaDoPaulo.saca(20);
        System.out.println(contaDoPaulo.saldo);
        System.out.println(conseguiuRetirar);

        Conta contaDaMarcela = new Conta();
        contaDaMarcela.deposita(1000);

        if(contaDaMarcela.transfere(300, contaDoPaulo)) {
            System.out.println("transferencia com sucesso");
        } else {
            System.out.println("faltou dinheiro");
        }

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

Caso a linha que inserimos o `if` fique muito grande ou complicada, é normal quebrarmos em duas linhas. Poderíamos alocar a parte do código referente ao método `transfere()` e inserir uma variável `sucessoTransferencia` em seu lugar.

```
public class TestaMetodo {
    public static void main(String[] args) {
        Conta contaDoPaulo = new Conta();
        contaDoPaulo.saldo = 100;
        contaDoPaulo.deposita(50);
        System.out.println(contaDoPaulo.saldo);

        boolean conseguiuRetirar = contaDoPaulo.saca(20);
        System.out.println(contaDoPaulo.saldo);
        System.out.println(conseguiuRetirar);

        Conta contaDaMarcela = new Conta();
        contaDaMarcela.deposita(1000);

        boolean sucessoTransferencia = contaDaMarcela.transfere(300, contaDoPaulo);
```

```
            if(sucessoTransferencia) {
                System.out.println("transferencia com sucesso");
            } else {
                System.out.println("faltou dinheiro");

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