01

Atributos privados e encapsulamento

Transcrição

Neste ponto do curso, já sabemos como trabalhar com métodos e atributos, inclusive, atributos que servem como referência para outros objetos. Veremos qual é o nosso novo fator limitante para o progresso do projeto **ByteBank**.

Criaremos uma nova classe de teste chamada `TesteSacaNegativo`.

Nosso objetivo é ficar com uma quantidade negativa de dinheiro. Usaremos o nome da variável `conta`, mas poderia ser qualquer outro de sua escolha, diferente do **tipo** `Conta`, que, obrigatoriamente, precisa ser o nome de uma classe que exista no sistema.

Lembrando que é muito comum criar uma variável que tenham o mesmo nome da classe. Depositaremos `100` reais na `conta`.

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
    }
}
```

Feito isso, tentaremos sacar `200` reais, e verificaremos o que acontece quando executamos a aplicação.

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
        conta.saca(200);
        System.out.println(conta.saldo);
    }
}
```

O resultado da aplicação foi a impressão do saldo `100`. Isso ocorreu, pois o saque não foi efetivado, já que o valor disponível na conta não era o suficiente.

Quando mantemos o "Ctrl" pressionado e passamos o mouse sobre a código, vemos que o método retornou o valor `false`. Podemos passar o valor booleano diretamente para o `sysout`, ou seja, não precisamos sempre guardar o retorno de um método dentro de uma variável.

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
        System.out.println(conta.saca(200));
        System.out.println(conta.saldo);
    }
}
```

Ao executarmos novamente aplicação, veremos o valor `false` impresso. Conseguimos estipular o critério de que *nenhuma conta pode ter valores negativos*.

Porém, existe um truque que tentaremos fazer. Ao tentarmos sacar `101` reais da nossa conta, cujo saldo é `100`, o procedimento não será efetivado. A não ser que, diretamente no atributo `saldo`, estipulamos que o seu valor é o `saldo - 101`.

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
        System.out.println(conta.saca(101));
        System.out.println(conta.saldo);

        conta.saldo = conta.saldo - 101;
        System.out.println(conta.saldo);
    }
}
```

No término da execução do programa, veremos que o valor impresso será `-1.0`. Supondo que seja um requisito do **ByteBank** que não haja valores negativos nas contas, temos um problema.

Uma solução é avisar para os funcionários do nosso banco para que nunca acessem o atributo `saldo` diretamente, e sim, invocando o método adequado.

O ideal é que sempre trabalhemos utilizando os **métodos**, nunca diretamente os atributos. Podemos utilizar a analogia do funcionamento de um carro: utilizamos o pedal de aceleração para que o carro se locomova, não precisamos abrir o capo e reviramos mecanismos de injeção de gasolina de forma manual para gerar a aceleração necessária.

A ideia é utilizar a interface adequada para a melhor funcionalidade de um sistema. A forma mecânica de funcionamento do carro está escondida, ou seja, *encapsulada*.

O que precisamos saber operar é a metodologia de funcionamento do carro, o mesmo vale para a conta bancária.

Queremos, portanto, que a manipulação direta de atributos seja proibida.

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
        System.out.println(conta.saca(200));
        System.out.println(conta.saldo);

        //proibido
        conta.saldo = conta.saldo - 101;
        System.out.println(conta.saldo);
    }
}
```

No Java, existe a possibilidade de ocultarmos um atributo, deixá-lo privado. Na classe `Conta`, escreveremos ao lado do atributo que queremos encapsular a palavra-chave `private`

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

    // ...
```

A partir do momento em que um atributo se torna **privado**, isso quer dizer que ele não pode ser lido ou modificado, a não ser na própria classe. Esse é o conceito principal de encapsulamento.

Ainda existe um problema: na classe `TesteSacaNegativo`, não podemos mais imprimir o valor de `saldo` através do `sysout`. É preciso utilizar um novo método para acessar o atributo `saldo`.

Na classe `Conta`, criaremos um método que devolve/informe o `saldo`. Chamaremos esse novo método de `pegaSaldo`, que não precisará receber parâmetro, mesmo assim, os parênteses são obrigatórios.

Ao lado esquerdo do método, colocaremos o seu retorno: um `double`. Escreveremos, também, o `public`. Dentro do método, diremos que ele simplesmente retorna o valor de `saldo`, utilizando a palavra-chave `return`.

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

    public void deposita(double valor) {...}

    public boolean saca(double valor) {...}

    public boolean transfere(double valor, Conta destino, Conta origem) {...}

    public double pegaSaldo() {
        return this.saldo;
    }
```

É interessante que as funções sejam bem localizadas no código, por isso, o objetivo desse método é simplificar os processos de manutenção do sistema sem que precisemos fazer alterações em múltiplos trechos do código.

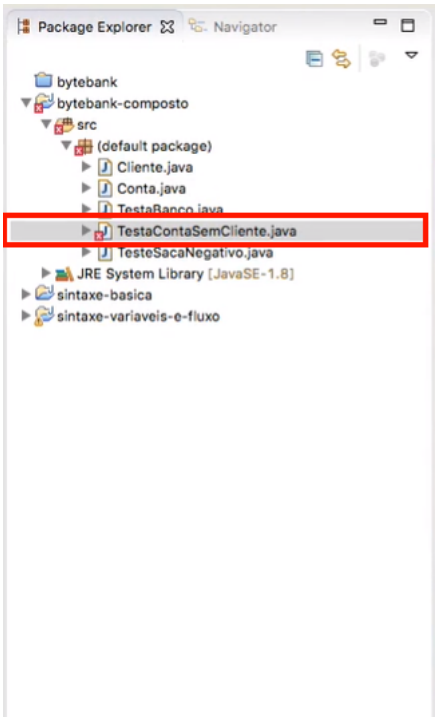
De volta a classe `TesteNegativo`, iremos invocar o método `pegaSaldo`:

```
public class TesteSacaNegativo {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.deposita(100);
        System.out.println(conta.saca(101));

        conta.saca(101);
```

```
        System.out.println(conta.pegaSaldo());
    }
}
```

Há uma classe no nosso projeto que ainda está acessando o atributo `saldo` da maneira antiga.



O `TestaContaSemCliente` está dessa forma:

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

        // ...
```

Vamos corrigir o código :

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

        // ...
```