01

Construtores

Transcrição

Chegamos ao ponto em que temos uma conta bem encapsulada, de forma que não é mais possível inserir um saldo negativo, pois ele está velado pelos métodos de manipulação da conta, como `saca()`, `transfere()` e `deposita()`.

Caso encontremos algum *bug*, como a inserção de um depósito negativo, basta adicionar um `if` no método `deposita()`. As soluções para possíveis problemas do nosso código estão bem localizadas, ou seja, alterado o código em um único ponto podemos realizar a manutenção do nosso sistema.

O próximo ponto que iremos nos voltar é se as nossas contas conseguem ser criadas de uma maneira que sempre possuam dados consistentes.

Quando pensamos em objetos consistentes, queremos dizer que seus atributos funcionam de acordo com as regras de negócios estipuladas por uma empresa, chefe ou algo do gênero.

Para testarmos a consistência dos objetos do nosso banco, criaremos mais uma classe chamada `TestaValores`. Sabemos que não é possível criar uma conta e deixá-la com um valor negativo, pois um saque não seria permitido neste caso e nem outros métodos de alteração de saldo seriam eficientes neste sentido. Entretanto podemos criar uma conta e atribuir ao número de `agencia` valores negativos.

```
public class TestaValores {
    public static void main(String[] args) {
        Conta conta = new Conta();
        conta.setAgencia(-50);
        conta.setNumero(-330);
    }
}
```

Na classe `Conta`, precisaremos adicionar `if`s nos métodos `setAgencia` e `setNumero`, afirmando que caso seja postulado um valor menor ou igual a zero, ocorrerá uma mensagem de erro e a execução será interrompida.

```
public class Conta {
    // atributos

    // método deposita
    // método saca
    // método tranfere
    // método pegaSaldo

    public int getNumero() {...}

    public void setNumero(int numero) {
        if (numero <= 0) {
            System.out.println("não pode valor <= 0");
            return;
        }
        this.numero = numero;
    }
```

```
    public int getAgencia() {...}

    public void setAgencia(int agencia) {
        if (agencia <= 0) {
            System.out.println("nao pode valor menor igual a 0");
            return;
        }
        this.agencia = agencia;
    }

    public void setTitular(Cliente titular) {...}

    public Cliente getTitular() {...}
}
```

Com isso, ao tentarmos executar o programa da classe `TestaValores`, que está com números negativos para o atributo `agencia`, será impressa a mensagem `nao pode valor menor igual a 0`.

Porém, não estamos totalmente protegidos desses valores negativos. Percebam o problema caso escrevamos a seguinte linha de código:

```
public class TestaValores {
    public static void main(String[] args) {
        conta.setAgencia(-50);
        conta.setNumero(-330);
        System.out.println(conta.getAgencia());
    }
}
```

Veremos como resultado da aplicação o valor impresso `0`. Portanto, não solucionamos o problema da numeração, porque estamos trabalhando com o valor *default*, lembrem-se que no momento em que acionamos o `new` em um objeto, os atributos são zerados, e os atributos de tipo referência recebem valor `null`.

Muitas vezes encontramos objetos que nascem em um estado inconsistente com relação à regra de negócio. Existe uma forma de restringirmos dados: toda a vez que criamos um objeto somos obrigados a passar informações específicas, fazemos isso através de um **construtor**.

Os parênteses `()` fechados ao lado do objeto `Conta` estão invocando um construtor.

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

        // ...
```

O construtor é um trecho de código caracterizado pela seguinte conformação:

```
public Conta() {

}
```

Não escrevemos esse código ao longo do curso. O Java automaticamente insere esse código, é o que chamamos de **construtor padrão**.

Iremos escrever esse código em nossa classe `Conta` e acionaremos o `sysout`.

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


    public Conta() {
        System.out.println("estou criando uma conta");

        // ...
}
```

Feito isso, executaremos a aplicação da classe `TestaValores`. Lembrando que o construtor está executando a terceira linha do código da classe.

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

        // ...
```

Veremos que o resultado impresso será `estou criando uma conta`.

Os parênteses estão passando um construtor. Não se trata de um método, o construtor não possui um retorno `void`, `double`, ele é uma rotina de inicialização.

> O construtor é executado apenas uma vez no momento em que construímos um objeto. Não há como executar duas vezes o construtor para um mesmo objeto.

O construtor nos oferece a possibilidade de inicializar alguns dados, como por exemplo, podemos estipular que o saldo inicial de uma conta vale `100` reais. Mas o mais interessante é que o construtor pode receber parâmetros.

Para que o construtor seja invocado na abertura de uma nova conta, é necessário obrigatoriamente que seja passada a agência dessa conta e seu número. Feito isso, os atributos podem ser populados. Na execução iremos exibir o número da conta, assim conseguimos ver a atuação do construtor.

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

    public Conta( int agencia, int numero) {
        this.agencia = agencia;
        this.numero = numero;
```

```
        System.out.println("estou criando uma conta" + this.numero);
    }

    // ...

}
```

O construtor padrão que o Java estipula caso não tenhamos escrito nenhum outro construtor, deixa de existir.

Através do construtor podemos definir restrições e exigir informações específicas do objeto. O que será exigido pelo construtor varia de acordo com as regras de negócio, no caso do nosso banco é interessante que o saldo inicie com zero, por exemplo.

Definimos alguns parâmetros para o construtor, então, para cada objeto criado, precisaremos primeiramente comunicar a `agencia` (`1337`) e o `numero` da conta (`24226`).

```
public class TestaValores {
    public static void main(String[] args) {
        Conta conta = new Conta(1337, 24226);

        // ...
```

Ao executarmos a aplicação, o resultado impresso será `estou criando uma conta 24226` .

Podemos adicionar `if` s no nosso código para gerar as condições ideais de criação do objeto de acordo com as regras de negócio estipuladas. Por exemplo, que o número de `agencia` deve ser maior ou igual a zero.

Com a presença do construtor e os parâmetros que a ele foram passados, não é mais necessário que haja no nosso código os métodos `setAgencia()` e `setNumero()` . Se for estipulado como regra de negócio que uma conta sempre terá o mesmo número e a mesma agência, não há necessidade de evoca métodos para alterá-la.

Trata-se de um **atributo imutável**, e há vários casos em que essa é uma opção interessante.

Uma dica de navegação no Eclipse: a nossa classe `Conta` está grande. Para facilitar o acesso ao conteúdo da classe, no cabeçalho do Eclipse, selecionaremos a opção "Window > Show View > Outline" (atalho "Ctrl + O"). Ao lado esquerdo da tela, surgirá um resumo de todos os métodos e atributos presentes na classe, podemos clicar em cada um deles para termos um acesso rápido.