4.12 EXERCÍCIOS: ORIENTAÇÃO A OBJETOS

O modelo da conta a seguir será utilizado para os exercícios dos próximos capítulos.

O objetivo aqui é criar um sistema para gerenciar as contas de um `Banco`. **Os exercícios desse capítulo são extremamente importantes**.

1. Modele uma conta. A ideia aqui é apenas modelar, isto é, só identifique que informações são importantes. Desenhe no papel tudo o que uma `Conta` tem e tudo o que ela faz. Ela deve ter o nome do titular (`String`), o número (`int`), a agência (`String`), o saldo (`double`) e uma data de abertura (`String`). Além disso, ela deve fazer as seguintes ações: saca, para retirar um valor do saldo; deposita, para adicionar um valor ao saldo; calculaRendimento, para devolver o rendimento mensal dessa conta.

2. Transforme o modelo acima em uma classe Java. Teste-a, usando uma outra classe que tenha o `main`. Você deve criar a classe da conta com o nome `Conta`, mas pode nomear como quiser a classe de testes, contudo, ela deve possuir o método `main`.

   A classe Conta deve conter pelo menos os seguintes métodos:

   - `saca` que recebe um `valor` como parâmetro e retira esse valor do saldo da conta
   - `deposita` que recebe um `valor` como parâmetro e adiciona esse valor ao saldo da conta
   - `calculaRendimento` que não recebe parâmetro algum e devolve o valor do saldo multiplicado por 0.1

   Um esboço da classe:

```
class Conta {

    double saldo;
    // seus outros atributos e métodos

    void saca(double valor) {
        // o que fazer aqui dentro?
    }



    void deposita(double valor) {
        // o que fazer aqui dentro?
    }

    double calculaRendimento() {
        // o que fazer aqui dentro?
    }
}
```

   Você pode (e deve) compilar seu arquivo java sem que você ainda tenha terminado sua classe `Conta`. Isso evitará que você receba dezenas de erros de compilação de uma vez só. Crie a classe `Conta`, coloque seus atributos e, antes de colocar qualquer método, compile o arquivo java. O arquivo `Conta.class` será gerado, mas não podemos "executá-lo" já que essa classe não tem um `main`. De qualquer forma, a vantagem é que assim verificamos que nossa classe `Conta` já está tomando forma e está escrita em sintaxe correta.

   Esse é um processo incremental. Procure desenvolver assim seus exercícios, para não descobrir só no fim do caminho que algo estava muito errado.

   Um esboço da classe que possui o `main`:

```
class TestaConta {

    public static void main(String[] args) {
        Conta c1 = new Conta();

        c1.titular = "Hugo";
        c1.numero = 123;
        c1.agencia = "45678-9";
```

TODAS AS CLASSES NO MESMO ARQUIVO?

Você até pode colocar todas as classes no mesmo arquivo e apenas compilar esse arquivo. Ele vai gerar um `.class` para cada classe presente nele.

Porém, por uma questão de organização, é boa prática criar um arquivo `.java` para cada classe. Em capítulos posteriores, veremos também determinados casos nos quais você será **obrigado** a declarar cada classe em um arquivo separado.

Essa separação não é importante nesse momento do aprendizado, mas se quiser ir praticando sem ter que compilar classe por classe, você pode dizer para o `javac` compilar todos os arquivos java de uma vez:

```
javac *.java
```

3. Crie um método `recuperaDadosParaImpressao()`, que não recebe parâmetro mas devolve o texto com todas as informações da nossa conta para efetuarmos a impressão.

   Dessa maneira, você não precisa ficar copiando e colando um monte de `System.out.println()` para cada mudança e teste que fizer com cada um de seus funcionários, você simplesmente vai fazer:

   ```
   Conta c1 = new Conta();
   // brincadeiras com c1....
   System.out.println(c1.recuperaDadosParaImpressao());
   ```

   Veremos mais a frente o método `toString`, que é uma solução muito mais elegante para mostrar a representação de um objeto como `String`, além de não jogar tudo pro `System.out` (só se você desejar).

   O esqueleto do método ficaria assim:

   ```
   class Conta {

       // seus outros atributos e métodos

       String recuperaDadosParaImpressao() {
           String dados = "Titular: " + this.titular;
           dados += "\nNúmero: " + this.numero;
           // imprimir aqui os outros atributos...
           // também pode imprimir this.calculaRendimento()
           return dados;
       }
   }
   ```

4. Construa duas contas com o `new` e compare-os com o `==`. E se eles tiverem os mesmos atributos? Para isso você vai precisar criar outra referência:

   ```
   Conta c1 = new Conta();
   ```

```
c1.titular = "Danilo";
c1.saldo = 100;

Conta c2 = new Conta();
c2.titular = "Danilo";
c2.saldo = 100;

if (c1 == c2) {
    System.out.println("iguais");
} else {
    System.out.println("diferentes");
}
```

5. Crie duas referências para a **mesma** conta, compare-os com o `==`. Tire suas conclusões. Para criar duas referências pra mesma conta:

```
Conta c1 = new Conta():
c1.titular = "Hugo";
c1.saldo = 100;

c2 = c1;
```

O que acontece com o `if` do exercício anterior?

6. (opcional) Em vez de utilizar uma `String` para representar a data, crie uma outra classe, chamada `Data`. Ela possui 3 campos `int`, para dia, mês e ano. Faça com que sua conta passe a usá-la. (é parecido com o último exemplo da explicação, em que a `Conta` passou a ter referência para um `Cliente`).

```
class Conta {
    Data dataDeAbertura; // qual é o valor default aqui?
    // seus outros atributos e métodos
}

class Data {
    int dia;
    int mes;
    int ano;
}
```

Modifique sua classe `TestaConta` para que você crie uma `Data` e atribua ela a `Conta`:

```
Conta c1 = new Conta();
//...
Data data = new Data(); // ligação!
c1.dataDeAbertura = data;
```

Faça o desenho do estado da memória quando criarmos um `Conta`.

7. (opcional) Modifique seu método `recuperaDadosParaImpressao` para que ele devolva o valor da `dataDeAbertura` daquela `Conta`:

```
class Conta {

    // seus outros atributos e métodos
    Data dataDeAbertura;

    String recuperaDadosParaImpressao() {
```

```
        String dados = "\nTitular: " + this.titular;
        // imprimir aqui os outros atributos...

        dados += "\nDia: " + this.dataDeAbertura.dia;
        dados += "\nMês: " + this.dataDeAbertura.mes;
        dados += "\nAno: " + this.dataDeAbertura.ano;
        return dados;
    }
}
```

Teste-o. O que acontece se chamarmos o método `recuperaDadosParaImpressao` antes de atribuirmos uma data para esta `Conta` ?

8. (opcional) O que acontece se você tentar acessar um atributo diretamente na classe? Como, por exemplo:

```
Conta.saldo = 1234;
```

Esse código faz sentido? E este:

```
Conta.calculaRendimento();
```

Faz sentido perguntar para o esquema da Conta seu valor anual?

9. (opcional-avançado) Crie um método na classe `Data` que devolva o valor formatado da data, isto é, devolva uma String com "dia/mes/ano". Isso para que o método `recuperaDadosParaImpressao` da classe `Conta` possa ficar assim:

```
class Conta {
    // atributos e metodos

    String recuperaDadosParaImpressao() {
        // imprime outros atributos...
        dados += "\nData de abertura: " + this.dataDeAbertura.formatada();
        return dados;
    }
}
```

1.10 DESAFIOS