Exercícios CONTA: orientação a objetos

O modelo da conta a seguir será utilizado em aula para e pode ser aplicado em outros exercícios.

O objetivo aqui é criar um sistema para gerenciar as contas de um `Banco`.

Modele uma conta. A ideia nesse momento é apenas modelar, isto é, identificar quais 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, que é de 10% sobre o saldo.
Resposta:
Modelando uma conta
Toda conta **tem**:

- String titular;
- int numero;
- String agencia;
- double saldo;
- String dataDeAbertura;

2. Toda conta **faz**:
   - `saca`: retira uma determinada quantia do saldo da conta;
   - `deposita`: adiciona uma determinada quantia ao saldo da conta;
   - `calculaRendimento`: devolve o quanto essa conta rende por mês.
3. Transforme o modelo acima em uma classe Java. Teste a classe 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. Por exemplo, pode chamá-la `TestaConta`. Contudo, ela deve necessariamente ter o método `main`.

A classe Conta deve conter, além dos atributos mencionados anteriormente, pelo menos os seguintes métodos:

- `saca`, que recebe um `valor` como parâmetro e o retira do saldo da conta;
- `deposita`, que recebe um `valor` como parâmetro e o adiciona 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 sacar(double valor) {
        saldo = saldo - VlrSaque
    }

    void depositar(double valor) {
        saldo = saldo + Vlrdeposito
    }

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

Você pode (e deve) compilar seu arquivo Java sem que ainda tenha terminado sua classe `Conta`. Isso evitará que você receba dezenas de erros de compilação de uma vez só. Crie a classe `Conta`, ponha seus atributos e, antes de colocar qualquer método, compile o arquivo Java. O arquivo `Conta.class` será gerado, mas não podemos executá-lo, visto que essa classe não tem um `main`. De qualquer

forma, verificamos, assim, que nossa classe `Conta` já está tomando forma e está escrita em sintaxe correta.
Esse é um processo incremental. Procure desenvolver seus exercícios assim para não descobrir só no fim do caminho que algo estava muito errado.
Um esboço da classe que tem o `main`:

```
   class TestaConta {

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

           c1.titular = "Hugo";
           c1.numero = 123;
           c1.agencia = "45678-9";
           c1.saldo = 50.0;
           c1.dataDeAbertura = "04/06/2015";

           c1.deposita(100.0);
           System.out.println("saldo atual:" + c1.saldo);
           System.out.println("rendimento mensal:" +
c1.calculaRendimento());
       }
   }
```

Incremente essa classe. Faça outros testes, imprima outros atributos e invoque os métodos que você criou a mais.
Lembre-se de seguir a convenção Java, isso é importantíssimo. Preste atenção nas maiúsculas e minúsculas, seguindo este exemplo: `nomeDeAtributo`, `nomeDeMetodo`, `nomeDeVariavel`, `NomeDeClasse`, etc.

Todas as classes no mesmo arquivo?

Você até pode colocar todas as classes no mesmo arquivo e compilar apenas esse arquivo. Ele gerará 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 praticar sem ter que compilar classe por classe, você pode dizer para o `javac` compilar todos os arquivos Java de uma vez:

```
javac *.java
```

Abaixo a resposta completa desse item:

```
class Conta {
    String titular;
    int numero;
    String agencia;
    double saldo;
    String dataDeAbertura;

    void saca (double valor) {
        this.saldo -= valor;
    }

    void deposita (double valor) {
        this.saldo += valor;
    }

    double calculaRendimento() {
        return this.saldo * 0.1;
    }
4.  }
```

Na classe `Conta`, 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 os seus funcionários, você simplesmente fará:

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

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

Veremos, mais à 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 para o `System.out` (somente se você o 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;
        }
    }
```

Abaixo está a resposta completa desse item:

```
    class Conta {
        // Outros atributos e métodos.

        String recuperaDadosParaImpressao() {
            String dados = "Titular: " + this.titular;
            dados += "\nNúmero: " + this.numero;
            dados += "\nAgência: " + this.agencia;
            dados += "\nSaldo: R$" + this.saldo;
            dados += "\nData de abertura: " + this.dataDeAbertura;
            return dados;
        }
    5.    }
```

Na classe de teste dentro do bloco `main`, construa duas contas com o `new` e compare-as com o `==`. E se elas tiverem os mesmos

atributos? Para isso, você precisará 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");
```