DANIEL ABELLA
www.daniel-abella.com

Daniel Abella

Profissional PMP, PMI-ACP e CSM, Gerente de Projetos no Virtus & UniFacisa e Professor do Curso de Sistemas de Informação da UniFacisa. Autor dos livros Gestão A3 (Ágil) e Scrum Arretado.

Conteúdo:

Capítulo 3

Orientação à Objetos

Livro Java do Zero (Uma Viagem ao Mundo Java)

01 Classes e Objetos

- Como uma linguagem OO, Java é amparada primordialmente por Classes.

IDENTIFICANDO CLASSES

Vamos entender como identificar classes dado um conjunto de informações do cliente com um exemplo prático.

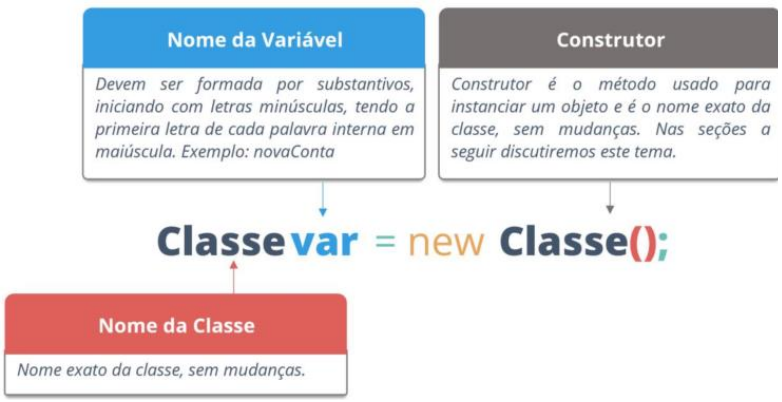
Imagine que, o cliente solicitou a criação de um aplicativo chamado AbellaBank que, possui uma Agencias e Contas. Uma Agência, possui o seu código, gerente responsável, cidade, endereço e uma lista de Contas, que por sua vez possui um número da agência, número da conta, nome do titular e saldo em conta.

Neste exemplo, podemos identificar duas classes: Agencia e Conta. Note que, as classes são geralmente no Singular e em Maiúsculo a primeira letra. A classe Agencia possui 5 variáveis de instância: código (int), gerenteResponsavel (String), cidade (String), endereço (String) e listaContas (Array ou ArrayList). Por fim, temos uma classe chamada Conta, que possui 4 variáveis de instância: numeroAgencia (int), numeroConta (int), titular (String) e saldo (double).

O mapeamento de classes e respectivos atributos foi unicamente baseado nas informações fornecidas pelo cliente (sublinhado acima).

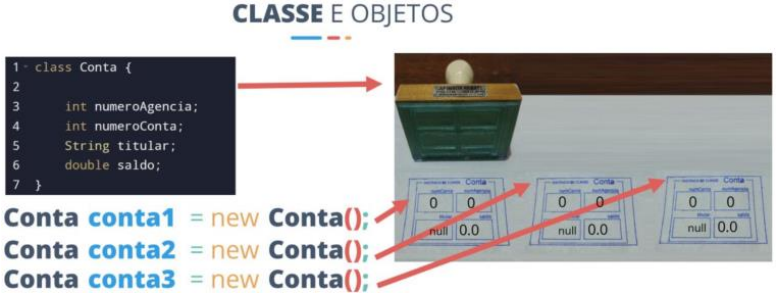
```java
class Conta {

    int numeroAgencia;
    int numeroConta;
    String titular;
    double saldo;
}
```

- O projeto de instanciar, isto é, gerar objetos, é apresentado a seguir.



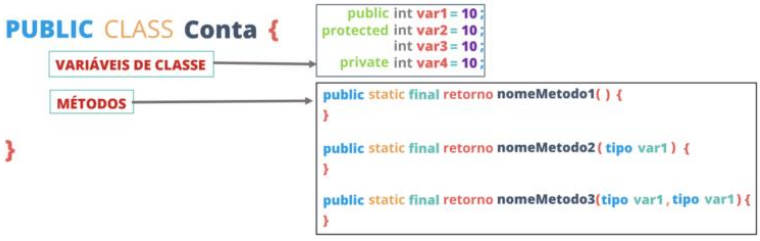
- Um exemplo da instanciação da classe Conta apresentada anteriormente é apresentado abaixo.

```java
class TesteInstaciacao {


    public static void main(String[] args) {
        Conta conta1 = new Conta();

        Conta conta2;
        conta2 = new Conta();
    }
}
```



02 Métodos

- Dentro das classes, podemos ter variáveis e métodos, sendo este último tema desta seção. Na classe Conta, a seguir, é possível declarar variáveis, bem como métodos. Não tem uma ordem específica, variáveis e depois métodos, pode ser o inverso, mas comumente por convenção declaramos as variáveis e logo depois os métodos.



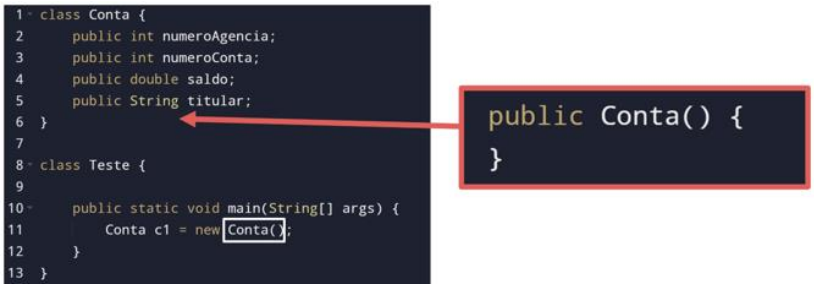

```java
public double getSaldoLiquido() {
    return saldo - 1.5;
}

public void depositar(double montante) {
    saldo = saldo + montante;
}
```

03 Métodos Construtores

- Construtores são métodos especiais usados para criar objetos a partir de uma classe.
- Abaixo temos três exemplos de construtores. Os construtores são iniciados por um modificador de acesso (public, protected, default ou private). O primeiro construtor possui modificador public, enquanto que o segundo possui modificador private. O último construtor possui o modificador default.

```
public Conta ( ) {
}

private Conta ( String var ) {
}

Conta (doublev1  doubel v2 ) {
}
```

- Quando uma classe não tem nenhum construtor explícito, o Java coloca implicitamente um construtor público vazio para você. Este construtor se chama Construtor Padrão ou Construtor Default e está representado à direita da imagem a seguir. Por isso o código da linha 11 funciona.



04 Sobrecarga de Métodos e Construtores

- Sobrecarga é a possibilidade de definirmos múltiplas versões de um método ou construtor com o mesmo nome, mas com diferentes tipos de parâmetros. Veja o exemplo deste conceito no código abaixo.

```
1  class Calculadora {
2
3      public double soma(double operando1, double operando2) {
4          return operando1 + operando2;
5      }
6
7      public double soma(double operando1, double operando2, double
           operando3) {
8          return operando1 + operando2 + operando3;
9      }
10 }
```

05 Encapsulamento

- Imagine o cenário a seguir. Note que, acessamos todas as variáveis da classe Conta sem nenhuma proteção. Ou seja, como exemplificado nas linhas 12 e 13 a seguir, podemos atribuir qualquer valor à variável saldo. Para proteger as variáveis, o remédio é encapsulamento.
- Para aplicar o remédio encapsulamento, temos 2 passos:
- #1 Restringir as variáveis: Ou seja, colocar modificadores de acesso mais restritos como private; e
- #2 Criar métodos que promovam a interação com a variável de modo seguro: No caso do saldo, podemos ter ao menos 3 métodos (depositar, sacar e getSaldo).
- O resultado é apresentado a seguir.

```
1  class Conta {
2      private int numeroAgencia;
3      private int numeroConta;
4      private double saldo;
5      private String titular;
6
7      public void depositar(double montante) {
8          if(montante > 0) {
9              saldo += montante;
10         }
11     }
12
13     public void sacar(double montante) {
14         if(montante > 0) {
15             saldo -= montante;
16         }
17     }
18
19     public double getSaldo() {
20         return this.saldo;
21     }
22 }
23
24 class Teste {
25
26     public static void main(String[] args) {
27         Conta c1 = new Conta();
28         c1.depositar(1000);
29         System.out.println("Saldo atual " + c1.getSaldo());
30
31         c1.sacar(300);
32         System.out.println("Saldo atual " + c1.getSaldo());
33     }
34 }
```

- O encapsulamento protege os dados (estado interno) e comportamentos de uma classe, sendo uma forma de ocultar a implementação interna da classe e fornece acesso controlado aos seus atributos e métodos.

06 Getters e Setters

- Os métodos getters e setters desempenham um papel fundamental na implementação do encapsulamento em Java. São métodos públicos usados para obter (get) e modificar (set) os valores dos atributos privados de uma classe. Podemos verificar no exemplo a seguir, a criação do getter e setter para o atributo titular. Note que, o método setTitular, ajusta o valor da variável titular (linha 5) com o valor do parâmetro titular da linha 7.

```
1  class Conta {
5      private String titular;
6
7      public void setTitular(String titular) {
8          this.titular = titular;
9      }
10
11     public String getTitular() {
12         return titular;
13     }
17 }
```

Gerar Getters e Setters

Ficar gerando getters e setters para todas as variáveis de instância tende a ser uma tarefa chata, porém necessária. Pensando nisto, temos 2 possibilidades.

A primeira é que, as IDEs como Eclipse ou Intellij Idea, podem gerar os getters e setters baseado nas variáveis declaradas, como veremos na seção 3 do capítulo 4.

A segunda possibilidade é usar bibliotecas externas como Lombok que, geram os getters e setters automaticamente. Na seção 4.1 do capítulo a seguir, apresentamos brevemente como usar esta biblioteca para geração.

07 Static

- Static é uma palavra-chave, aplicável em atributos ou métodos, que faz com que este membro pertence à classe em si, em vez de pertencer a uma instância específica da classe.

Métodos Static

- Imagine um cenário que fosse necessário um método para tratar uma String, removendo os espaços e transformando em maiúsculo. Neste cenário, este método não faz sentido estar em uma classe como Conta.
- Podemos colocar métodos como o supracitado como static, de modo que, não precisa mais instanciar a classe para chama-lo, basta apenas chamada da seguinte maneira: Classe.metodoStatic().

```
1  class StringUtil {
2
3      public static String tratarString(String textoSemFormato) {
4          textoSemFormato = textoSemFormato.trim();
5          textoSemFormato = textoSemFormato.toUpperCase();
6          return textoSemFormato;
7      }
8  }
9
```

```
10 class Teste {
11     public static void main(String[] args) {
12         String strFormatada = StringUtil.tratarString(" daniel ");
13         System.out.println(strFormatada);
14     }
15 }
```

Variáveis Static

- Uma variável de instância vira variável de classe quando recebe a palavra-chave static.

```
1  class Carro {
2
3      private String descricao;
4      private int ano;
5      public static int quantidadeVendas;
6
7      public Carro(String descricao, int ano) {
8          this.descricao = descricao;
9          this.ano = ano;
10         quantidadeVendas++;
11     }
12
13     public static int mostrarQuantidadeVendas() {
14         return quantidadeVendas;
15     }
16 }
```

- Constantes são variáveis que seu valor não pode ser modificado. Imagine uma constante chamada MAIORIDADE_PENAL com valor 18, cujo valor não pode ser nunca mais alterado. Na linguagem Java, não existe formalmente o conceito de constantes, mas combinamos as palavras-chave static e final para termos algo similar. Constantes em Java é a combinação das palavras-chave static e final. Sugere-se que, constantes devem sempre ser em maiúscula com palavras internas separadas por um underline.
- Final em variáveis significa que, uma vez o valor atribuído, não pode ser alterado.

```
public static final double PI = 3.14159;
public static final int MAX_ATTEMPTS = 5;
public static final String ERROR_MESSAGE =
        "Ocorreu um erro inesperado.";
```

- Agora, a partir de qualquer classe, se precisássemos imprimir o valor de PI, por exemplo, basta fazer: System.out.println(ExemploConstantes.PI);