

[< VOLTAR](#)

No definition found.

[Search the web for "encapsulamento" »](#)

# Encapsulamento

Encapsulamento é um termo que se refere a proteção de acesso aos atributos, métodos e classes. Nesta aula vamos ver como proteger métodos e atributos e qual a diferença entre o public, private e protected, no Java. Veremos, também, a implementação dos famosos "setters" e "getters" de parâmetros.

## NESTE TÓPICO

- Encapsulamento. O que é?
- Visibilidade em Java (modificadores de acesso)
- Sets e gets
- Resumo da aula
- Referências



## Encapsulamento. O que é?

A palavra encapsulamento tem a ver com capsula, em português. Para explicar o conceito de encapsulamento, vamos fazer a analogia ao conceito de capsula.

Para que serve uma capsula espacial (daquelas que a Nasa manda para a lua com alguém dentro)? Bem, a princípio, para proteger o indivíduo que vai lá dentro do mundo externo e vice-versa, certo?

E que tal uma capsula de remédio, o que é? Bem, é um tipo de ?gelatina? (que o corpo consegue absorver pois é digerível) que protege o medicamento que está dentro dele (normalmente em pó) do mundo externo. Imagine se os remédios que vêm em capsulas viessem em pó e tivéssemos que pegar a medida exata antes de ingerir. Seria incabível. Então a capsula, neste caso, serve também para outra coisa: para fornecer quem a utiliza a quantidade certa de medicamento.



Capsulas de remédio, contém o componente ativo dentro da capsula de gelatina

Bem, o conceito de **encapsulamento** em Java não é muito diferente. Até o momento nós programamos com classes públicas (`public class NomeDaClasse`), certo? Trabalhamos também com métodos públicos, por exemplo: `public float somar(float a, float b)`.



É hora de começarmos a proteger nossas classes e que sejam acessadas somente por quem deve acessá-la, evitando-se, assim, violações de acesso e melhorando a organização do código.

Mas por que proteger meu código, ou por que não posso deixar tudo público? Bem, responda mentalmente a seguinte pergunta: "Qual a cor do cavalo branco de Napoleão?". Intuitivamente você diz que é branco, pois a resposta está na pergunta. Agora pergunta-se: No mundo orientado a objetos, quem sabe a cor do cavalo? Muitos respondem "Napoleão?", mas não, quem sabe a cor do cavalo, no mundo orientado a objetos é o próprio cavalo, pois a característica (atributo) de cor está **associada** ao cavalo e não ao Napoleão. Então, quem sabe a cor do cavalo? No mundo OO, o próprio cavalo.

Isso quer dizer que devemos proteger nossos objetos de nós mesmos, ou seja, cuidar para que as responsabilidades sejam corretamente divididas dentro do conceito de orientação a objetos. Em outras palavras, a orientação a objetos requer que tudo esteja em seu devido lugar e seja acessado apenas quando necessário.

Antes de colocarmos a mão na massa, é preciso conhecer as opções de visibilidade que Java nos proporciona.

## Visibilidade em Java (modificadores de acesso)

Segundo Teruel, na declaração de um método, variável ou constante, deve-se indicar de onde podem ser acessados, se apenas de dentro da própria classe (*private*), se de qualquer classe da aplicação (*public*), ou se apenas de dentro do pacote ao qual a classe pertence ou de subclasses ligadas em uma relação de herança (*protected*).

São os modificadores de visibilidade quem definem se uma classe, método, atributo ou constante poderão (e como poderão) ser acessados. A tabela abaixo mostra a relação destes modificadores de acesso com o nível do código (acesso na mesma classe, acesso no mesmo pacote, acesso em subclasses e acesso global - todas as classes):

Modificador	Mesma classe	Mesmo pacote	Subclasses	Todas as classes
private	Sim			
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	
sem modificador	Sim	Sim		

Modificadores de acesso em Java



Em resumo, podemos dizer também, sobre os modificadores de acesso que (Teruel):

- **public:** Permite que a classe, o método ou o atributo seja acessado a partir de qualquer ponto da aplicação, independente do pacote.
- **private:** Somente pode ser acessado dentro da classe e serve apenas para métodos e variáveis. Classes não podem ser private pois não poderiam ser acessadas.
- **protected:** O modificador protected permite que o recurso seja acessado de dentro da mesma classe, de qualquer classe que esteja no mesmo pacote da classe que contém o recurso e de subclasses da classe que contém o recurso em uma relação de herança
- **sem modificador (default):** Em Java, dizermos que atributos, classes e métodos sem modificadores estão com a visibilidade default. Na ausência de um modificador, o recurso poderá ser acessado apenas de dentro da própria classe e de classes que estão no mesmo pacote.

É muito importante conhecer os modificadores de acesso em Java e saber quais são suas características, ou seja, para que serve cada um.

## Sets e gets

Por padrão de boas práticas de programação, a partir de agora todos os atributos locais de uma classe serão privados (private), ou seja, somente a própria classe terá acesso aos seus atributos, para que nenhum objeto externo o modifique deliberadamente.

Mas então, como os atributos receberão valores do mundo externo? Muito simples: Através dos métodos get e set.

Como os próprios nomes dos métodos sugerem, cada um deles possui uma função e para cada um dos atributos em sua classe, você terá UM método set para ele e UM método get para ele.

- O método **set** serve para que o valor desse atributo seja alterado pelo mundo externo ao objeto; O método set deve receber um atributo do mesmo tipo que a variável local que será setada. O método set é sempre void (sem retorno).
- O método **get** serve para que o mundo externo a seu objeto receba o valor deste atributo. O método get não recebe nenhum parâmetro, mas é tipado, ou seja, ele deve ser do mesmo tipo que a variável que está sendo retornada.

Para exemplificar, vamos criar uma pequena calculadora com quatro operações, utilizando encapsulamento. Isso quer dizer que teremos uma classe chamada “Calculadora”, com dois atributos encapsulados e os métodos de soma, subtração, multiplicação e divisão e outra classe sendo a principal do projeto.

Veja a codificação da classe “Calculadora” abaixo. Repare nos comentários e repare, também, que para cada variável existe um método **get** e um método **set**.



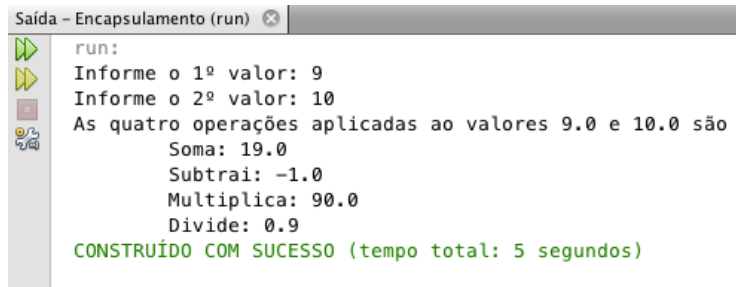
```
1.  public class Calculadora {
2.
3.      //Atributos locais privados
4.      private float a, b;
5.
6.      //get do A. Repare que ele tem o mesmo TIPO do a (float)
7.      public float getA() {
8.          return a;
9.      }
10.
11.     //set do A. Repare que ele recebe um float no parâmetro
12.     public void setA(float a) {
13.         this.a = a;
14.     }
15.
16.     //get do B
17.     public float getB() {
18.         return b;
19.     }
20.
21.     //set do B
22.     public void setB(float b) {
23.         this.b = b;
24.     }
25.
26.     //repare que o método não recebe nenhum atributo
27.     //pois ele usará os locais, que devem ser
28.     //setados antes pela classe que chama esta
29.     public float soma() {
30.         return a + b;
31.     }
32.
33.     public float subtrai() {
34.         return a - b;
35.     }
36.
37.     public float multiplica() {
38.         return a * b;
39.     }
40.
41.     public float divide() {
42.         try {
43.             return a / b;
44.         } catch (Exception e) {
45.             System.out.println(e.getMessage());
46.             return 0;
47.         }
48.     }
49. }
```



Agora veja a classe principal, logo abaixo. Note que o acesso externo (no objeto da classe Principal) agora às variáveis que serão operadas ocorre através dos métodos **set**.

```
1. import java.util.Scanner;
2.
3. public class Principal {
4.
5.     public static void main(String args[]) {
6.         float numero1, numero2;
7.         Scanner sc = new Scanner(System.in);
8.         Calculadora calc = new Calculadora();
9.
10.        System.out.print("Informe o 1º valor: ");
11.        numero1 = sc.nextFloat();
12.
13.        System.out.print("Informe o 2º valor: ");
14.        numero2 = sc.nextFloat();
15.
16.        //Seta na instância de Calculadora (calc) os valores a e b
17.        calc.setA(numero1);
18.        calc.setB(numero2);
19.
20.        System.out.println("As quatro operações aplicadas ao valores " + calc.getA()
+ " e " + calc.getB() + " são");
21.        System.out.println("\tSoma: " + calc.soma());
22.        System.out.println("\tSubtrai: " + calc.subtrai());
23.        System.out.println("\tMultiplica: " + calc.multiplica());
24.        System.out.println("\tDivide: " + calc.divide());
25.    }
26. }
```

E um possível resultado deste programa pode ser visto abaixo:



```
Saída - Encapsulamento (run)
run:
Informe o 1º valor: 9
Informe o 2º valor: 10
As quatro operações aplicadas ao valores 9.0 e 10.0 são
    Soma: 19.0
    Subtrai: -1.0
    Multiplica: 90.0
    Divide: 0.9
CONSTRUÍDO COM SUCESSO (tempo total: 5 segundos)
```



#### Execução do projeto de encapsulamento

Em resumo o programa é similar a como sempre foi feito, contudo, agora os campos estão encapsulados corretamente, conforme imperativo das [boas práticas de programação](#).

Uma dica importante é que o NetBeans pode encapsular os campos automaticamente para você, com alguns cliques, ou seja, ele gera todos os códigos dos sets e gets automaticamente. Basta clicar com o botão direito na classe e clicar em “Refatorar” e “Encapsular campos”. Veja o vídeo abaixo do procedimento:

./videos/370386745.mp4



## Resumo da aula

Chegamos ao final de um importante conceito. Nesta aula você aprendeu que:

- É muito importante encapsular os atributos de uma classe, por boas práticas de programação e por padrões internacionais de desenvolvimento de aplicações.
- A visibilidade de classes, métodos e atributos pode ser alterada com as palavras reservadas public, protected, private e sem nenhum modificador de acesso na frente.
- Para encapsular é preciso seguir três passos:
  - Tornar os atributos privados
  - Criar um get para cada atributo
  - Criar um set para cada atributo
- Os métodos get são métodos de retorno de valor, então são do mesmo tipo do atributo e não recebem parâmetros
- Os métodos set são void (vazios) e não retornam nada, apenas alteram o valor do parâmetro. Para isso, ele deve receber um valor do mesmo tipo do atributo que está sendo modificado.



Pratique bastante encapsulando os campos de projetos que você já desenvolveu. Bons estudos e boa programação!

## Quiz

Exercício Final

Encapsulamento

INICIAR ➤

## Referências

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman



Avalie este tópico



ANTERIOR

Abstração de dados



Índice

Biblioteca

(https://www.uninove.br/conheca-

a-

uninove/biblioteca/sobre-

a-

biblioteca/apresentacao/)

Portal Uninove

(http://www.uninove.br)

Mapa do Site

Ajuda?

(https://ava.un  
idCurso=)

Organização em Pa

© Todos os direitos reservados

