

# Programação Orientada a Objetos

## Aula 05 - Encapsulamento

# Apresentação

---

Nesta aula, vamos aprender sobre encapsulamento, você sabe o que é isso? Cápsula nos lembra qualquer forma pequena que protege algo em seu interior, como um medicamento, fruto seco ou mesmo um compartimento para astronautas cheio de instrumentos para uma missão espacial. Você verá na aula de hoje uma importante característica da Programação Orientada a Objetos, o encapsulamento, que nos ajuda a desenvolver programas com maior qualidade e flexibilidade para mudanças futuras.



## **Vídeo 01** - Apresentação

## Objetivos

Ao final desta aula, você será capaz de:

- Conhecer os conceitos ligados a encapsulamento;
- Entender, através de um contra-exemplo (exemplo errado), a importância do encapsulamento;
- Entender com a “correção” do exemplo como se aplica essa característica de encapsulamento, tão importante para a programação OO.

# Encapsulamento

---

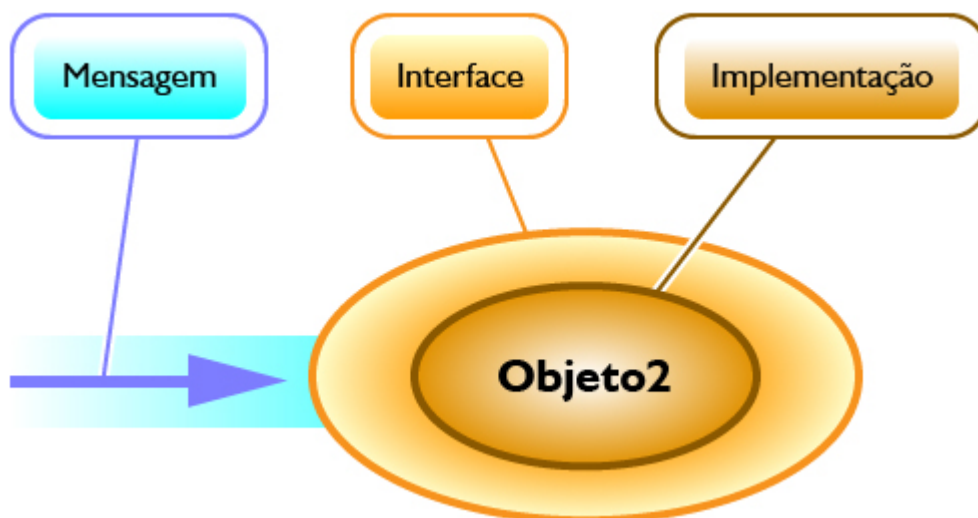
O que nos lembra a palavra encapsular? O que você diria se alguém pedisse a definição da palavra encapsulamento? E cápsula, lembra alguma coisa?

Encapsulamento é a característica da OO capaz de ocultar partes (dados e detalhes), de implementação interna de classes, do mundo exterior.

Graças ao encapsulamento, podemos ver as classes apenas pelos serviços (métodos) que elas devem oferecer para quem as utiliza. Não visualizamos, nesse caso, de que forma (como) o serviço (método) está implementado internamente na classe. No fundo, o encapsulamento da classe acaba definindo um contrato que determina o que o mundo exterior pode fazer com objetos daquela classe.

Na Figura 1, podemos ver que um objeto que possui a característica de **encapsulamento** fica protegido por uma cápsula. Essa cápsula, que chamamos de **interface**, serve para ocultar e proteger de outros objetos, os detalhes de implementação daquele objeto. Dessa maneira, o objeto só disponibiliza, através da interface, os serviços ou funcionalidades que ele deseja receber [mensagens](#) ([Veja mais sobre Troca de Mensagens na aula 4 – Sistemas OO.](#)) (solicitações) de outros objetos.

**Figura 01** - Um objeto com encapsulamento



Para facilitar nosso entendimento, vamos a um exemplo. Considere um objeto **Automóvel** que disponibiliza para um objeto **Pessoa** (na OO tudo é objeto!) a direção como parte da sua interface, para que se possa guiar o **Automóvel** para a esquerda ou para a direita. Através da direção, a **Pessoa** solicita ao **Automóvel** esses serviços, sem saber **COMO** serão feitos ou estão implementados.

Apenas o **Automóvel** sabe que mecanismos serão acionados para atender a solicitação da **Pessoa**. A **Pessoa** apenas usufrui do resultado da solicitação. Observe a Figura 2.

**Figura 02** - Pessoa usando a direção como interface para acessar os serviços de Automóvel



Podemos dizer, nesse caso, que estamos aplicando a característica de **encapsulamento** ao objeto **Automóvel**, pois ocultamos do objeto **Pessoa** os detalhes da implementação dos serviços oferecidos pelo Automóvel.

Você deve estar se perguntando: como isso acontece na prática?

Antes de mostrarmos a implementação na linguagem Java do uso do encapsulamento, precisamos conhecer os chamados **Modificadores de Acesso** da linguagem.



### Vídeo 02 - Encapsulamento

## Antes, uma Explicação...

---

Quando falarmos em **ACESSO**, é necessário interpretarmos: o acesso deve-se dar **a quem e por quem?**

Para respondermos, devemos interpretar que o acesso será ao objeto que possui encapsulamento (ou está encapsulado), e será feito pelos outros objetos do sistema ou mesmo por você (programador), que decidirá usar determinados objetos durante a construção de seus programas.

Outra pergunta seria: acesso **a que?** Para responder, devemos interpretar como o acesso aos atributos e métodos do objeto encapsulado.

Então, **ACESSO** nos traz a interpretação de acessar os atributos e métodos do objeto que possui encapsulamento pelos outros objetos do sistema ou pelos programadores.

## Modificadores de Acesso

---

Os **Modificadores de Acesso** são palavras-chave ou reservadas da linguagem Java cuja utilidade é permitir ou proibir o acesso aos atributos e/ou métodos das classes. Veja-os a seguir.

**public:** garante que o atributo ou método da classe seja acessado ou executado a partir de qualquer outra classe.

**private:** pode ser acessado, modificado ou executado apenas por métodos da mesma classe, sendo totalmente oculto ao programador (ou outros objetos do sistema) que for usar instâncias dessa classe.

**protected:** funciona como o **private**, exceto que as classes filhas ou derivadas também terão acesso ao atributo ou método. Veremos mais sobre classes filhas na aula de Herança.

**Package ou Friendly:** não são palavras reservadas de modificadores de acesso. Os atributos e métodos são chamados de Package ou Friendly, quando não possuem modificadores, ou seja, são os atributos e métodos declarados sem modificadores. Isso significa que podem ser acessados por todas as classes pertencentes a um mesmo pacote (pacotes são pastas onde estão inseridos os arquivos das classes, para ajudar a organizá-las. Falaremos sobre pacotes em aulas futuras).



### Vídeo 03 - Modificadores de Acesso

## Atividade 01

---

1. Relembrando os exemplos apresentados até agora, escolha 3 classes apresentadas e decida quais atributos e métodos seriam mais convenientes para serem declarados como privados e quais poderiam ser públicos.

# Um Exemplo Completo

---

Agora que você já conhece os conceitos sobre encapsulamento, vamos ver como isso acontece na prática.

Iremos mostrar um exemplo que demonstre a importância de se usar o encapsulamento na Programação Orientada a Objetos.

Vamos supor que temos uma classe **Agenda** que guarda apenas uma data e uma anotação importante a ser lembrada, conforme mostra a listagem 1 a seguir.

```
1 class Agenda{
2     int dia;
3     int mes;
4     String anotacao;
5 }
```

**Listagem 1** - Classe agenda

Agora, vamos querer que essa classe anote a informação. Assim, nossa classe fica:

```
1 class Agenda{
2     int dia;
3     int mes;
4     String anotacao;
5
6     void anote(int d, int m, String nota){
7         dia = d;
8         mes = m;
9         anotacao = nota;
10    }
11 }
```

**Listagem 2** - Adicionando comportamento

É importante também verificar se a data da anotação é uma data válida. Se for uma data válida, registra-se a anotação, caso contrário a anotação recebe uma informação de data inválida. Assim, nossa classe fica:

```

1 class Agenda{
2     int dia;
3     int mes;
4     String anotacao;
5
6     void anote(int d, int m, String nota){
7         dia = d;
8         mes = m;
9         anotacao = nota;
10
11         validaData();
12     }
13     void validaData(){
14         if ((dia<1) || (dia>31) || (mes<1) || (mes>12)){
15             dia = 0;
16             mes = 0;
17             anotacao = "Anotação não inserida devido a data inválida";
18         }
19     }
20     void mostraAnotacao(){
21         System.out.println(dia+"/"+mes" : "+ anotacao);
22     }
23 }

```

### Listagem 3 - Validação das datas

Você até agora pode está se perguntando onde está o encapsulamento e aquela conversa toda do início da aula, não é mesmo?

Calma! O objetivo desse exemplo é mostrar a fragilidade do código que **não usa o encapsulamento**. Por isso, iremos agora testar nossa classe **Agenda** com uma aplicação.

Para isso, vamos criar dois objetos com a classe Agenda: **agenda1** e **agenda2**. Em seguida, daremos a **agenda1** – uma anotação com uma data válida, e a **agenda2** – uma data inválida. Finalmente, consultaremos os dados dos atributos de cada objeto, em especial a anotação que foi registrada, utilizando o método **mostraAnotacao()**.



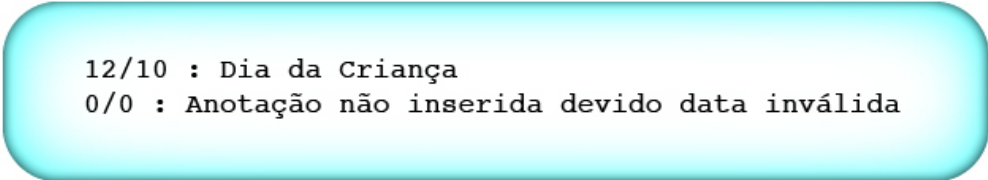
```
1 public class Principal{
2     public static void main(String[] args){
3         Agenda agenda1 = new Agenda();
4         Agenda agenda2 = new Agenda();
5
6         agenda1.anote(12,10,"Dia da Criança");
7         agenda2.anote(7,15,"Independência do Brasil");
8
9         agenda1.mostraAnotacao();
10        agenda2.mostraAnotacao();
11    }
12 }
```

**Listagem 4** - Testando a agenda

Para a **agenda1**, anotamos que dia 12 do mês 10 (Outubro) é o Dia da Criança, já para a **agenda2** anotamos que dia 7 do mês 15 (mês Inexistente) é o dia da Independência do Brasil.

Com a sua atual experiência de programador, observando a classe Agenda e os dados inseridos pela classe Principal, o que você acha que será impresso após termos rodado a aplicação? Vejamos então o resultado.

**Figura 03** - Saída impressa no terminal



```
12/10 : Dia da Criança
0/0 : Anotação não inserida devido data inválida
```

Perfeito!!! Não utilizamos o encapsulamento na classe Agenda e tudo funcionou perfeitamente! Sem nenhuma via de acesso desprotegida, correto? Errado! Vamos identificar a falha da nossa codificação.

Veja o que acontece se fizéssemos uma pequena modificação no método **main()**, observe a listagem 5 a seguir.

```

1 public class Principal{
2     public static void main(String[] args){
3         Agenda agenda1 = new Agenda();
4         Agenda agenda2 = new Agenda();
5
6         agenda1.anote(12,10,"Dia da Criança");
7         agenda2.anote(7,15,"Independência do Brasil");
8
9         agenda1.mostraAnotacao();
10        agenda2.mostraAnotacao();
11
12        agenda2.dia=7;
13        agenda2.mes = 15;
14        agenda2.anotacao = "Independência do Brasil";
15
16        agenda2.mostraAnotacao();
17    }
18 }

```

**Listagem 5** - Modificando o método main

O resultado dessa aplicação seria...

```

12/10 : Dia da Criança
7/15 : Independência do Brasil

```

Veja que sua Agenda permitiu que você inserisse uma data inválida para uma anotação. Isso significa que seu código está suscetível a falhas. Em um programa simples como esse, isso não irá trazer nenhuma dor de cabeça. Agora, imagine em um programa real usado diariamente pelas pessoas.

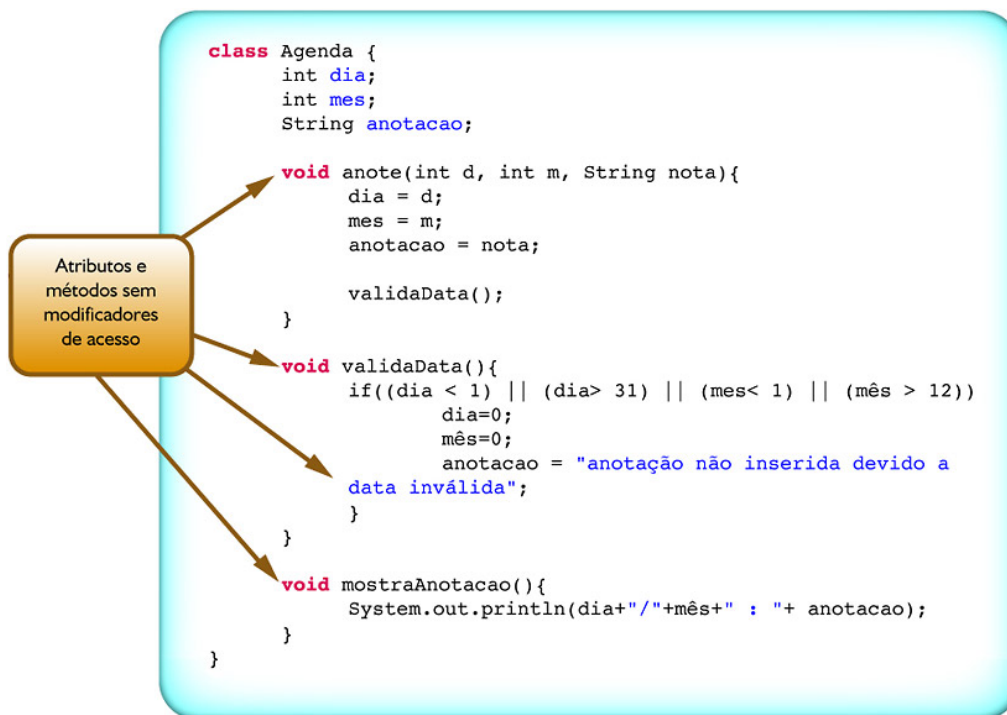
Qual é a solução? Respondendo: **a solução é aplicar o encapsulamento.**

## Aplicando o Encapsulamento ao Exemplo

---

Codificar em OO usando o encapsulamento nada mais é que aplicar os modificadores de acesso às classes.

Vamos ver como está a nossa classe Agenda com relação aos modificadores.

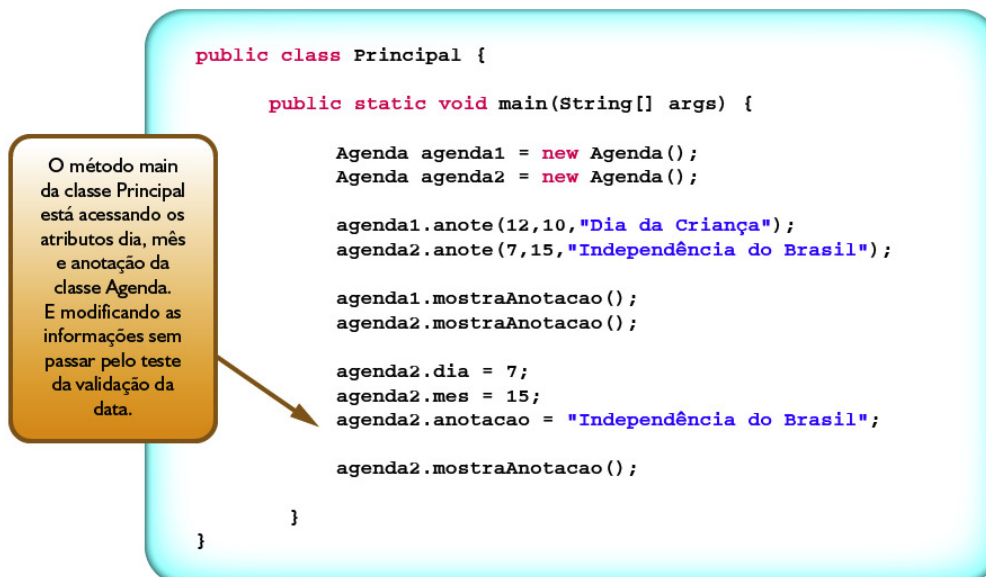


**Listagem 6** - Exemplo sem modificadores de acesso explícito

A classe **Agenda** não usa nenhum dos modificadores de acesso de forma explícita. Pela nossa classificação, implicitamente ele está usando o modificador de acesso Package ou Friendly. Vimos, anteriormente, que o modificador de acesso Package permite que classes do mesmo pacote acessem atributos ou métodos que estejam com tal modificador. Considerando que a nossa classe Principal está no mesmo pacote (veja sobre pacotes na classificação de Package), então, ele tem acesso aos atributos e métodos da classe Agenda.

Você pode estar se perguntando... Quando a classe Principal teve acesso aos atributos da classe Agenda?

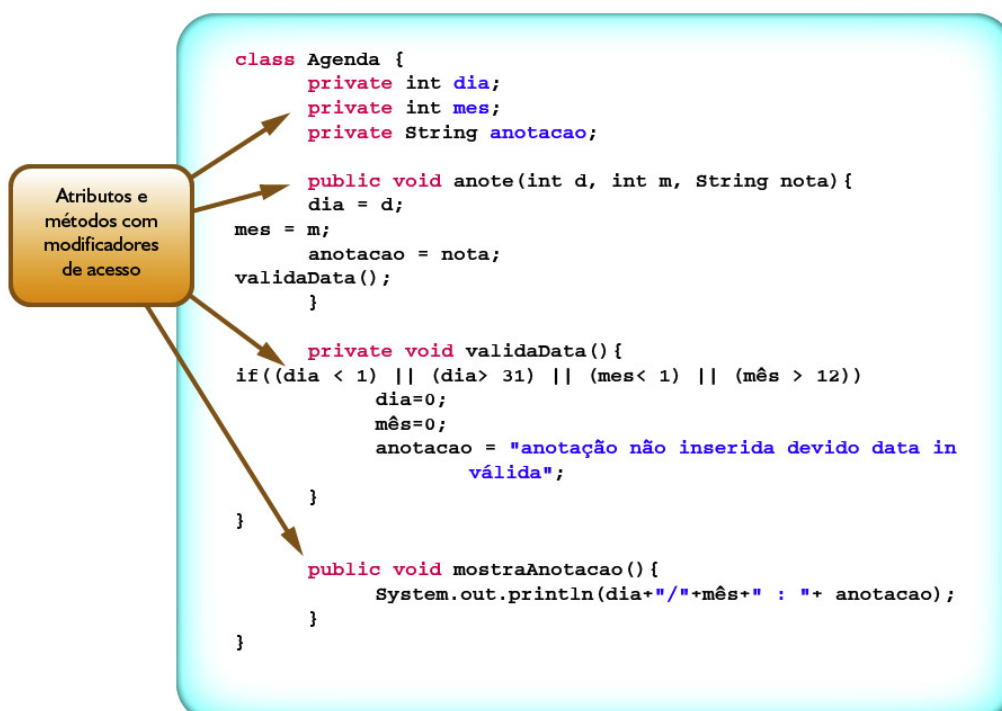
**Respondendo:** veja o trecho de código da classe Principal a seguir.



**Listagem 7** - Acesso direto aos métodos e atributos da classe

O método main() da classe Principal está acessando os atributos dia, mes e anotacao da classe Agenda. E modificando as informações sem passar pelo teste da validação da data. Essa falha na codificação não é aceitável.

Vamos então aplicar o encapsulamento tornando privados os atributos e o método validaData() da classe Agenda com o uso do modificador de acesso private, e permitir os acessos para os métodos anote() e mostraAnotacao(). Temos:



### Listagem 8 - Adicionando explicitamente os modificadores de acesso

Agora, o compilador já não aceita usar os comandos abaixo. Isso causaria erro! Ou seja, os atributos da classe Agenda agora são privados apenas para a uso da própria classe Agenda, como mostra a listagem 9.

Estes comandos causarão erros! Eles não são mais permitidos porque os atributos da classe Agenda agora são privados.

```
public class Principal {  
  
    public static void main(String[] args) {  
  
        Agenda agenda1 = new Agenda();  
        Agenda agenda2 = new Agenda();  
  
        agenda1.anote(12,10,"Dia da Criança");  
        agenda2.anote(7,15,"Independência do Brasil");  
  
        agenda1.mostraAnotacao();  
        agenda2.mostraAnotacao();  
  
        agenda2.dia = 7;  ERRO!  
        agenda2.mes = 15; ERRO!  
        agenda2.anotacao = "Independência do Brasil"; ERRO!  
  
        agenda2.mostraAnotacao();  
  
    }  
}
```

### Listagem 9 - Encapsulamento em ação

Com essa modificação, só é possível inserir uma anotação na classe Agenda usando o método anote(). O método anote() garante que a data inserida para a anotação será validada com o método validaData(), que também é privado ao uso apenas da classe Agenda.



**Vídeo 04** - Encapsulamento na Prática

Uma verdade que não pode ser omitida para os programadores: **você pode optar em não usar a característica de encapsulamento em seus códigos.**

Mas, isso seria uma **péssima prática** e um sinal que você não é um programador que segue as boas práticas. Se você havia pensado nisso, reveja os conceitos e tente entender a importância do encapsulamento para seus códigos.

## Atividade 02

---

1. Continuando a atividade anterior, separe as classes **Pessoa** e **Carro** e adicione os modificadores de acesso de seus atributos e métodos, verifique e corrija os programas para que não tentem executar uma operação não permitida pelo encapsulamento.

# Leitura Complementar

---

Segue uma recomendação de vídeo no youtube.  
<<https://www.youtube.com/watch?v=PsLCCBjzxlQ>>. Acesso em 07 set. 2017

## Resumo

---

Nesta aula, você viu os conceitos que envolvem o encapsulamento e sua importância para a programação OO. Os conceitos foram inicialmente ilustrados através de um exemplo que não aplicava o encapsulamento, mas, como consequência, trazia falhas ou erros ao desenvolvimento. Em seguida, aplicamos o encapsulamento ao exemplo e pudemos perceber que ele protegeu a classe Agenda, não permitindo modificações diretas e indevidas em seus atributos. Nesta aula, você não estudou a aplicação do modificador de acesso **protected** porque ele envolve o conceito de herança entre classe, o qual será abordado posteriormente. Esse então fica para uma das próximas aulas!

## Autoavaliação

---

1. Sem consultar o material responda: O que você entende por encapsulamento? Para que serve? E como aplicar?
2. Antes de rever os conceitos, diga para que servem os modificadores de acesso:
  - a. public
  - b. private
  - c. package
3. Pense e responda o que faria você decidir se um método deve ser usado como privado.

4. Crie classes chamadas Usuario e Hacker. Hacker possui o método main(). A classe Usuario possui os atributos *login* e *senha*. Inicialmente, não use encapsulamento e faça com que no método main() de Hacker seja possível modificar as informações (login, senha), inicialmente definidas, de um objeto da classe Usuário que você mesmo criar. Em seguida, aplique encapsulamento e verifique que Hacker terá suas tentativas frustradas.

## Referências

---

DEITEL, H. M.; DEITEL, P. J. **Java como programar**. Porto Alegre: Bookman, 2003.