
Unidade Acadêmica: Faculdade de Computação – FACOM
**Disciplina:** Programação Orientada a Objetos I
**Professor:** Fabiano Azevedo Dorça

# Prática 03

Objetivo:

- Aplicar a técnica de **encapsulamento** no sistema de gestão acadêmica

A técnica de encapsulamento visa esconder os detalhes de implementação de uma classe (atributos/métodos) do mundo externo, simplificando sua utilização e integração ao sistema.

Uma boa vantagem de se utilizar o encapsulamento é a proteção dos atributos da classe de valores indevidos, inválidos.

De acordo com o exposto em sala de aula, aplicando-se a técnica de encapsulamento à classe "Funcionário", o código desta classe seria alterado para:

```
import java.util.Scanner;

public class Funcionario
{
    private String nome;
    private String endereco;
    private String telefone;
    private String cpf;
    private String ctps;
    private double salario;

    public boolean setNome(String nome)
    {
        if (nome.length() > 0) {
            this.nome = nome;
            return true;
        }
        else  {
            System.out.println("Nome invalido!");
            return false;
        }
    }
```

```
public boolean setEndereco(String endereco)
{
    if (endereco.length() > 0){
        this.endereco = endereco;
        return true;
    }
    else {
        System.out.println("Endereço inválido!");
        return false;
    }
}

public boolean setTelefone(String telefone)
{
    if (telefone.length() > 0) {
        this.telefone = telefone;
        return true;
    }
    else {
        System.out.println("Telefone inválido!");
        return false;
    }
}

public boolean setCpf(String cpf)
{
    if (cpf.length() > 0) {
        this.cpf = cpf;
        return true;
    }
    else {
        System.out.println("CPF inválido!");
        return false;
    }
}

public boolean setCtps(String ctps)
{
    if (ctps.length() > 0) {
        this.ctps = ctps;
        return true;
    }
    else {
        System.out.println("CTPS inválido!");
        return false;
    }
}

public boolean setSalario(double salario)
{
    if (salario > 0) {
        this.salario = salario;
```

```
            return true;
        }
        else {
            System.out.println("Salario inválido!");
            return false;
        }
    }


    public void lerDados() {
       Scanner s = new Scanner(System.in);

       System.out.println("Digite o nome do funcionário:");
       while (!setNome(s.nextLine()));

       System.out.println("Digite o endereço do funcionário:");
       while (!setEndereco(s.nextLine()));

       System.out.println("Digite o telefone do funcionário:");
       while (!setTelefone(s.nextLine()));

       System.out.println("Digite o CPF do funcionário:");
       while (!setCpf(s.nextLine()));

       System.out.println("Digite a CTPS do funcionário:");
       while (!setCtps(s.nextLine()));

       System.out.println("Digite o salário do funcionário:");
       while (!setSalario(s.nextDouble()));

    }

    public void mostrarDados(){
        System.out.println("Nome: "+this.nome);
        System.out.println("Endereço: "+this.endereco);
        System.out.println("Telefone: "+this.telefone);
        System.out.println("CPF: "+this.cpf);
        System.out.println("CTPS: "+this.ctps);
        System.out.println("Salario: "+this.salario);
    }

}
```

Observe que todos os atributos da classe agora são privados e métodos específicos para leitura destes valores foram criados. Tais métodos, chamados de métodos “set”, são responsáveis por os valores de entrada e permitir ou não sua atribuição. **De qualquer forma, o acesso direto ao atributo jamais é permitido.**

**O método “lerDados()” foi agora alterado para utilizar os métodos “set”.**

**Os métodos "set" retornam `true` caso a atribuição tenha sido realizada com sucesso e `false` caso contrário.**

Observe que o método set é executado dentro de um laço `while` que é repetido enquanto o método "set" retornar false (atribuição inválida). Não se esqueça de que o operador "!" utilizado no laço é o operador lógico de negação (not).

## *Agora, Pede-se:*

- Implemente a técnica de encapsulamento **em todo** o seu sistema de gestão acadêmica.

- Para validação de cpf, ao invés de apenas testar se o cpf tem tamanho maior que zero, implemente o algoritmo real de validação de cpf (que é encontrado facilmente na Internet, implementado nas mais diversas linguagens).

- Crie um atributo "situacao" na classe aluno, para armazenar a situação do aluno no curso. O usuário deverá digitar:
  - M para matriculado
  - T para trancado
  - D para desligado

  Este atributo deve ser do tipo byte e deve armazenar:
  - 1 se o usuário digitar M
  - 2 se o usuário digitar T
  - 3 se o usuário digitar D

  Na hora de exibir os dados do aluno (mostrar dados), deve ser exibido: Matriculado, Trancado ou Desligado, de acordo com o código armazenado.

Observação sobre encapsulamento: Note que não é possível saber como este atributo "situacao" está implementado, ou seja, o tipo do atributo.

As demais classes do sistema apenas tem acesso à interface que dá acesso a este atributo, sem manipula-lo diretamente.

Com isto, obtemos transparência em relação à implementação interna das classes, seus tipos de dados, etc. A transparência de implementação deve ser sempre perseguida na programação orientada a objetos.

Para fazer comparação de strings é necessário utilizar o método `equals` da classe `String`, já que o comparador "==" funciona apenas para tipos primitivos (int, char, byte.....)

**O método `equals` recebe como parâmetro uma string e retorna `true` se forem iguais e `false` se forem diferentes.**

Exemplo:

```
String st = "teste";
if (st.equals("tes"))
    System.out.println("Strings são iguais.");
else
    System.out.println("Strings são diferentes.");
```

Outro método interessante da classe String é o método charAt. Este método recebe como parametro um inteiro de 0 a n-1, em que n é o tamanho da string, e retorna o caracter na posição n. O caracter é retornado no tipo char.

Exemplo:

```
String st = "Teste";
char c;
byte situacao;
c = st.charAt(0);
switch (c) {
    case 'M':
        situacao = 1;
```

```
        return true;
    case 'T':
        situacao = 2;
        return true;
    case 'D':
        situacao = 3;
        return true;
    default:
        return false;
}
```

**Para realizar comparação de caracteres, utilizamos o comparador de igualdade para tipos primitivos “==”**

Exemplo:

```
char c = 'a';
char d = 'b';
if (c == d)
    System.out.println(“Caracteres são iguais.”);
else
    System.out.println(“Caracteres são
diferentes.”);
```

---