

Algoritmos e Estruturas de Dados I
Aula07

Encapsulamento

Prof. MSc. Adalto Selau Sparremerger

 assparremerger@senacrs.com.br

  @adaltoss
 

 /assparremerger

Encapsulamento

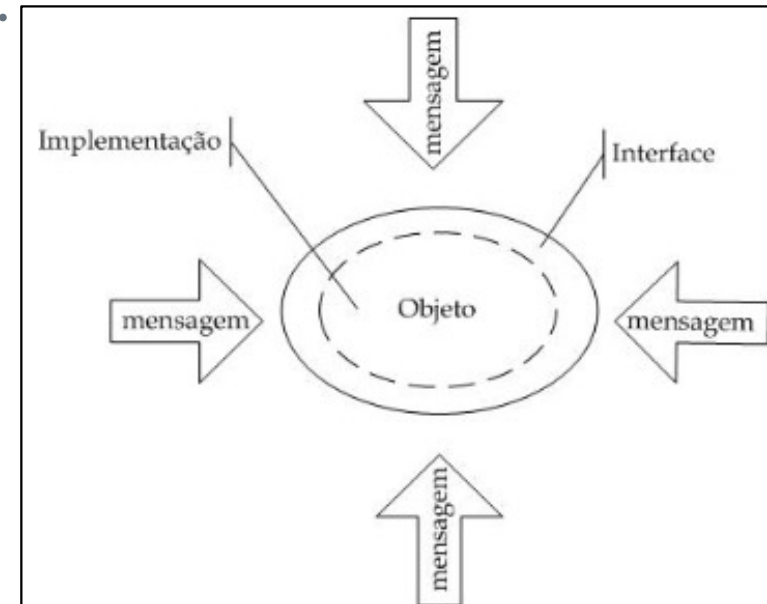
- ▷ Objetos possuem comportamento.
- ▷ O termo comportamento diz respeito a que operações são realizadas por um objeto e também de que modo estas operações são executadas.
- ▷ De acordo com o encapsulamento, objetos devem “esconder” a sua complexidade... Esse princípio aumenta qualidade em termos de:
 - Legibilidade
 - Clareza
 - Reuso

Encapsulamento

- ▷ O encapsulamento é uma forma de restringir o acesso ao comportamento interno de um objeto.
 - Um objeto que precise da colaboração de outro para realizar alguma tarefa simplesmente envia uma mensagem a este último.
 - O método (maneira de fazer) que o objeto requisitado usa para realizar a tarefa não é conhecido dos objetos requisitantes.
- ▷ Na terminologia da orientação a objetos, diz-se que um objeto possui uma interface.
 - A interface de um objeto é o que ele conhece e o que ele sabe fazer, sem descrever como o objeto conhece ou faz.
 - A interface de um objeto define os serviços que ele pode realizar e consequentemente as mensagens que ele recebe.

Encapsulamento

- ▷ Uma interface pode ter várias formas de implementação.
- ▷ Mas, pelo princípio do encapsulamento, a implementação utilizada por um objeto receptor de uma mensagem não importa para um objeto remetente da mesma.



(BEZERRA, 2007)

Encapsulamento

- ▷ Técnica que faz com que detalhes internos do funcionamento dos métodos de uma classe permaneçam ocultos para os objetos.
- ▷ Encapsular seria o mesmo que esconder todos os membros de uma classe, além de esconder como funcionam as rotinas (no caso métodos) do nosso sistema. Seria uma espécie de proteção.
- ▷ Encapsular é fundamental para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada.

Por que encapsular?

- ▷ A manutenção é favorecida pois, uma vez aplicado o encapsulamento, quando o funcionamento de um objeto deve ser alterado, em geral, basta modificar a classe do mesmo.
- ▷ O desenvolvimento é favorecido pois, uma vez aplicado o encapsulamento, conseguimos determinar precisamente as responsabilidades de cada classe da aplicação.

Modificadores de acesso

- ▷ Público (Public)
 - Qualquer classe tem acesso ao atributo ou método
- ▷ Protegido (Protected)
 - Apenas classes filhas (subclasses) tem acesso ao atributo ou método
- ▷ Privado (Private)
 - O atributo ou método só pode ser acessado dentro da própria classe

Métodos acessores e modificadores

- ▷ Geralmente, os campos de dados privados são de natureza técnica e interessam apenas ao criador das operações.
- ▷ Assim, o acesso aos atributos deve ser permitido pela implementação por meio de três itens:
 - Um campo de dados privado
 - Um método de leitura (acessador)
 - Um método de alteração (modificador)
- ▷ Por convenção, assessores e modificadores são chamados Getter and Setter
- ▷ Vantagens: Implementação interna pode ser modificada sem afetar nenhum código fora da própria classe.
- ▷ Os métodos “modificadores” podem fazer testes contra erros.

Modificadores de acesso em Python

- ▷ Existem, SQN!
- ▷ Filosofia Python: "somos todos adultos concordantes aqui"
- ▷ Há o conceito de **fracamente privado**, usando um underline (`_nomeAtributoFracamentePrivado`) no início dos atributos, apenas para sinalizar aos demais programadores que aquele atributo não deve ser acessado de fora de sua classe.
- ▷ Há também o conceito de **fortemente privado**, usando dois underlines (`__nomeAtributoFortementePrivado`) no início dos atributos, para sinalizar aos demais programadores que aquele atributo não pode ser acessado de fora de sua classe.

Getters

- ▷ Nomeamos um método acessor com GET toda vez que este método for verificar algum campo ou atributo de uma classe.
- ▷ Como este método irá verificar um valor, ele sempre terá um retorno. Mas não terá nenhum argumento.

▷ Em Java

```
public String getNome(){  
    return this.nome;  
}
```

▷ Em Python

```
def getNome(self):  
    return self.nome
```

Setters

- ▷ Nomeamos um método modificador com **set** toda vez que este método for modificar algum campo ou atributo de uma classe, ou seja, se não criarmos um método modificador **set** para algum atributo, isso quer dizer que este atributo não deve ser modificado.

▷ Em Java

```
public void setName(String nome){  
    this.nome = nome  
}
```

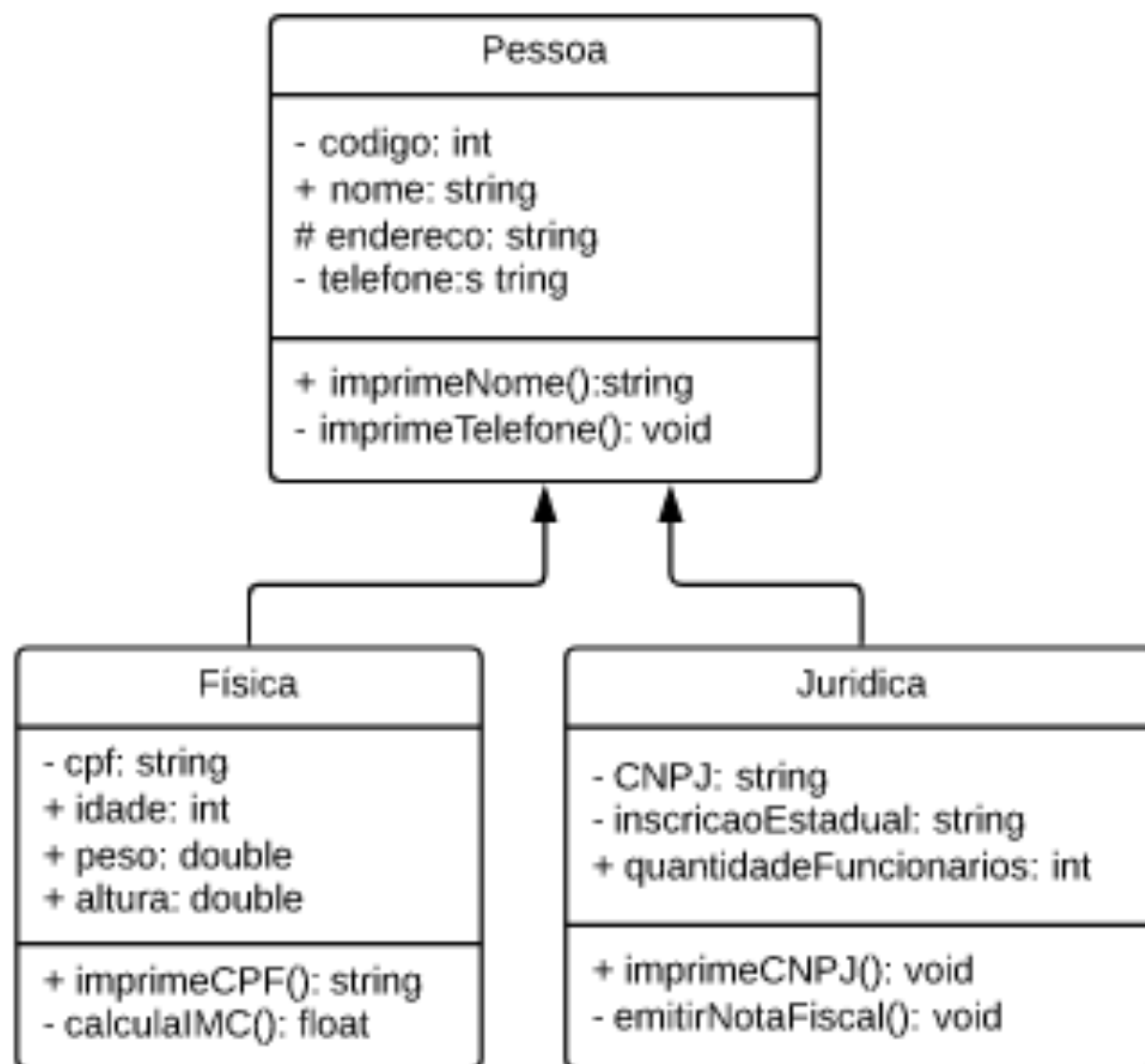
▷ Em Python

```
def setName(self, nome):  
    self.nome = nome
```

Métodos Privados

▷ O papel de alguns métodos pode ser o de auxiliar outros métodos da mesma classe. E muitas vezes, não é correto chamar esses métodos auxiliares de fora da sua classe diretamente.

```
1 class Conta:
2     def __init__(self):
3         self.saldo = 0.0
4
5     def __descontarTarifa(self):
6         self.saldo -= 1.99
7
8     def depositar( self, valor):
9         self.saldo += valor;
10        self.__descontarTarifa()
11
12    def sacar( self, valor):
13        self.saldo -= valor;
14        self.__descontarTarifa()
15
```



Referências

- ▷ BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. Rio de Janeiro: Elsevier, 2007.
- ▷ W3Schools:
<https://www.w3schools.com/python/default.asp>

Exercício

- ▷ Implementar as classes do diagrama a seguir, considerando os modificadores de acesso de cada atributo e método.

