



# Linguagem de Programação II

## Aula 08 – Encapsulamento

---

Paulo Vinicius Vieira

paulo.vieira@faculdadeimpacta.com.br

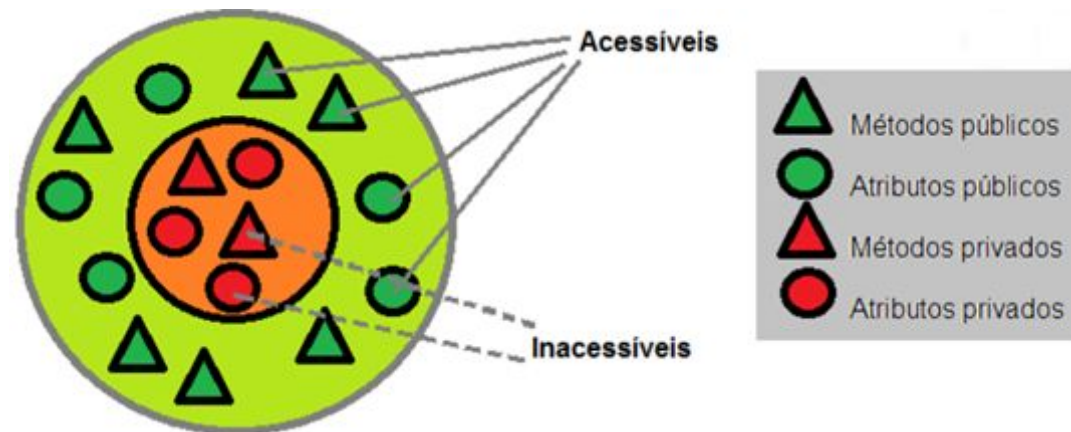
# Pilares da POO

---

- A POO se baseia em 4 pilares:
  - **Abstração**
    - Processo de representar entidades do mundo real
  - **Encapsulamento**
    - Adiciona segurança à aplicação
  - **Herança**
    - Facilita o reuso do código
  - **Polimorfismo**
    - Adicionam a possibilidade de alteração no funcionamento interno de objetos

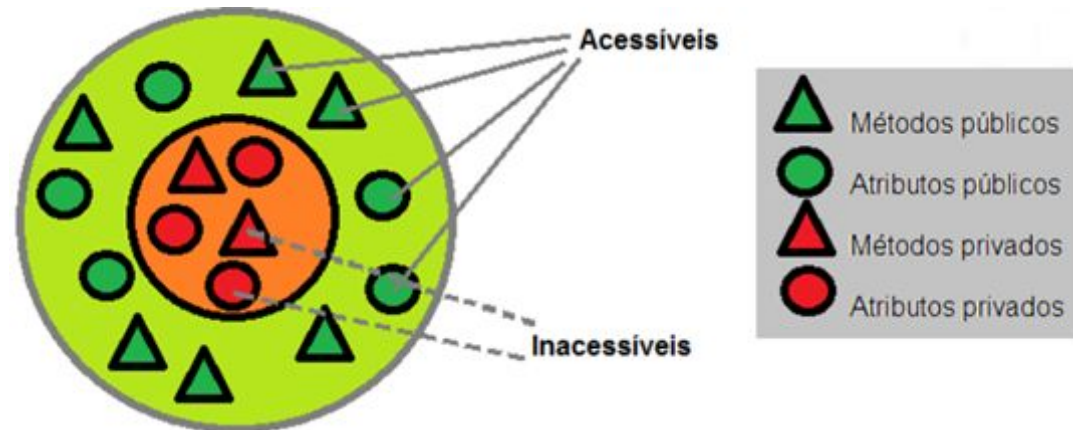
# Encapsulamento

- Encapsulamento diz respeito à proteção dos atributos ou métodos de uma classe
- Consiste em separar aspectos externos de um objeto dos detalhes internos de implementação
- Evita que dados específicos de uma aplicação possa ser acessado diretamente



# Encapsulamento

- Em Python, ao aplicar o conceito de encapsulamento, os atributos e métodos podem ser:
  - **Públicos**
  - **Privados**



# Encapsulamento

---

- Atributos ou métodos com nomes iniciados por dois sublinhados são **privados** e todas as outras formas são **públicas**

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg
```

# Encapsulamento

- Atributos ou métodos com nomes iniciados por dois sublinhados são **privados** e todas as outras formas são **públicas**

Atributos privados  
começam com dois  
sublinhados \_\_

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg
```

# Encapsulamento

- Os atributos ou métodos **privados** não podem ser acessados fora da classe.

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg

pessoa1 = Pessoa("João", 25, 11111111, 3333333)
pessoa1.nome = "João Paulo"
pessoa1.idade = 26
print(pessoa1.nome)
print(pessoa1.__cpf) ..... # ERRO
```

# Encapsulamento

Os atributos ou métodos **privados** só podem ser acessados dentro da classe.

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg

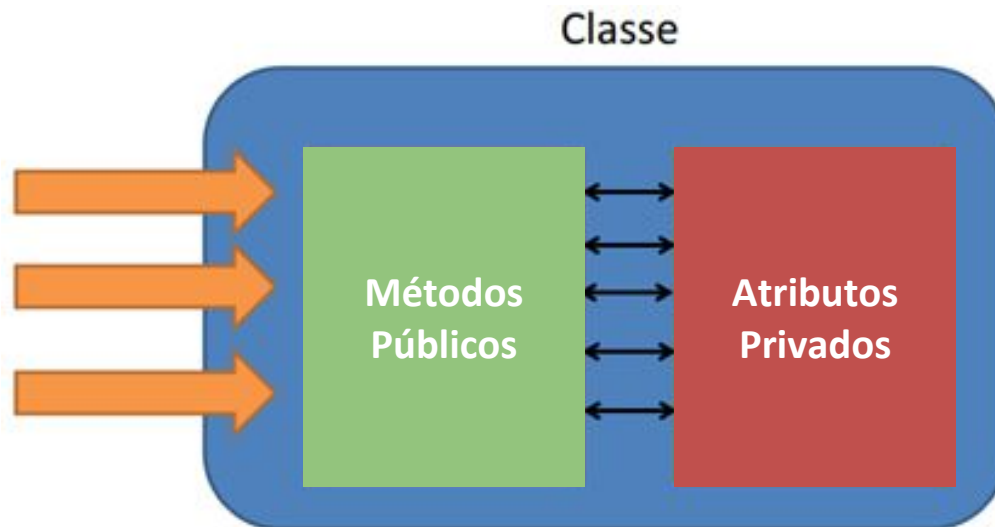
    def exibir_dados(self):
        print("Nome:", self.nome)
        print("Idade:", self.idade)
        print("CPF:", self.__cpf)
        print("RG:", self.__rg)

pessoa1 = Pessoa("João", 25, 11111111, 3333333)
pessoa1.nome = "João Paulo"
pessoa1.idade = 26
pessoa1.exibir_dados()
```



# Encapsulamento

- Aplicando o conceito de encapsulamento, os atributos e métodos privados de uma classe ficam ocultos do restante da aplicação
- Para permitir o acesso é necessário criar métodos públicos que acessem os atributos privados



# Encapsulamento

---

- a prática mais comum é criar dois métodos:
  - um que retorna o valor do atributo (**get**)
  - e outro que altera o valor do atributo (**set**)
- A convenção para esses métodos em muitas linguagens orientadas a objetos é colocar a palavra get ou set antes do nome do atributo.

# Encapsulamento

## Métodos *get*

- retorna valor do atributo

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg

    def get_cpf(self):
        return self.__cpf

    def get_rg(self):
        return self.__rg

    def set_cpf(self, cpf):
        self.__cpf = cpf

    def set_rg(self, rg):
        self.__rg = rg

pessoa1 = Pessoa("João", 25, 111111111, 3333333)
pessoa1.nome = "João Paulo"
pessoa1.idade = 26
pessoa1.set_cpf(22222222)
print("CPF:", pessoa1.get_cpf())
```

# Encapsulamento

## Métodos *get*

- retorna valor do atributo

## Métodos *set*

- Altera o valor do atributo

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg

    def get_cpf(self):
        return self.__cpf

    def get_rg(self):
        return self.__rg

    def set_cpf(self, cpf):
        self.__cpf = cpf

    def set_rg(self, rg):
        self.__rg = rg

pessoa1 = Pessoa("João", 25, 11111111, 3333333)
pessoa1.nome = "João Paulo"
pessoa1.idade = 26
pessoa1.set_cpf(2222222)
print("CPF:", pessoa1.get_cpf())
```

# Encapsulamento

Métodos *get* e *set* podem ser utilizados para realizar validação de acesso

```
class Pessoa:
    def __init__(self, nome, idade, cpf, rg):
        self.nome = nome
        self.idade = idade
        self.__cpf = cpf
        self.__rg = rg

    def get_cpf(self):
        return self.__cpf

    def get_rg(self):
        return self.__rg

    def set_cpf(self, cpf):
        if len(str(cpf)) == 11:
            self.__cpf = cpf
        else:
            print("Valor Inválido")

    def set_rg(self, rg):
        self.__rg = rg
```