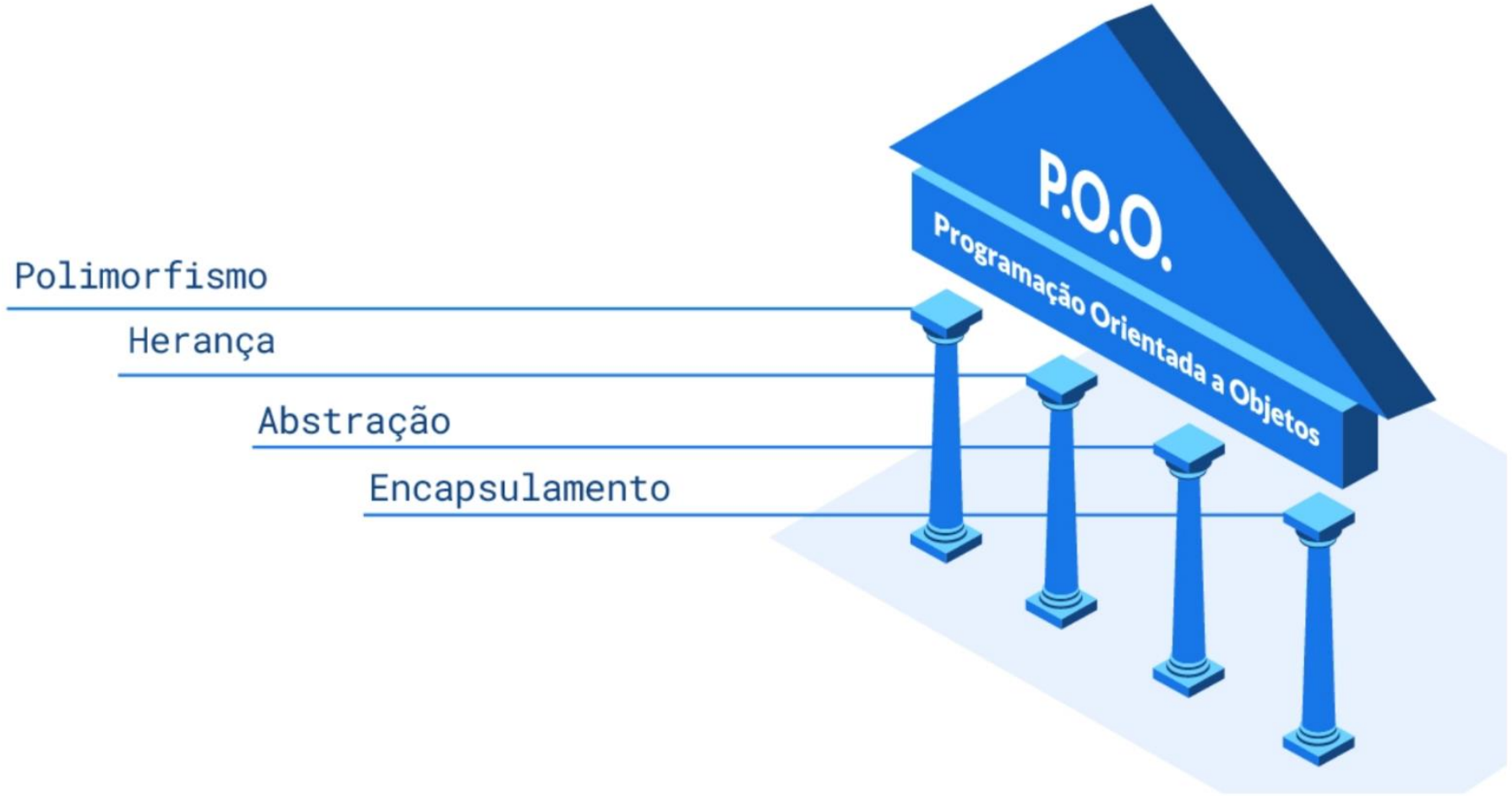
F<BR/C> DE SOFTWARE

UC: PROGRAMAÇÃO ESTRUTURADA E ORIENTADA A OBJETOS

PROFESSOR: THIAGO ALMEIDA

PILARES DA ORIENTAÇÃO A OBJETOS



ENCAPSULAMENTO

- Encapsulamento é a proteção dos atributos ou métodos de uma classe.
- Em Python existem somente o ***public*** e o ***private*** e eles são definidos no próprio nome do atributo ou método.
- Atributos ou métodos iniciados por no máximo dois sublinhados (underline) são privados e todas as outras formas são públicas.
- Após ***encapsular*** os atributos, eles só serão acessados via métodos ***getters e setters.***

MODIFICADORES DE ACESSO / VISIBILIDADE

- Encapsulamento Visibilidade
- A visibilidade é usada para indicar como uma determinada propriedade ou método poderá ser acessado. Há três formas possíveis: Público, protegido ou privado.
- (+) Public ou Público: Indica que a propriedade ou método, pode ser acessado por qualquer outra classe
- (#) Protected ou Protegido: Indica que a propriedade ou método, pode ser acessado pela classe e pelas classes derivadas. Classes filhos, por ex.
- (-) Private ou Privado: Indica que a propriedade ou método, pode ser acessado apenas pela classe.

ENCAPSULAMENTO

- O código de impressão abaixo irá retornar um erro:

```
class Retangulo:

    def __init__(self, lado_a, lado_b):
        self.__lado_a = lado_a
        self.__lado_b = lado_b
        print("Criada uma nova instância Retangulo")

    def get_lado_a(self):
        return self.__lado_a

    def calcula_area(self):
        return self.__lado_a * self.__lado_b

ret1 = Retangulo(8,12)
print(ret1.__lado_a)
```

ENCAPSULAMENTO

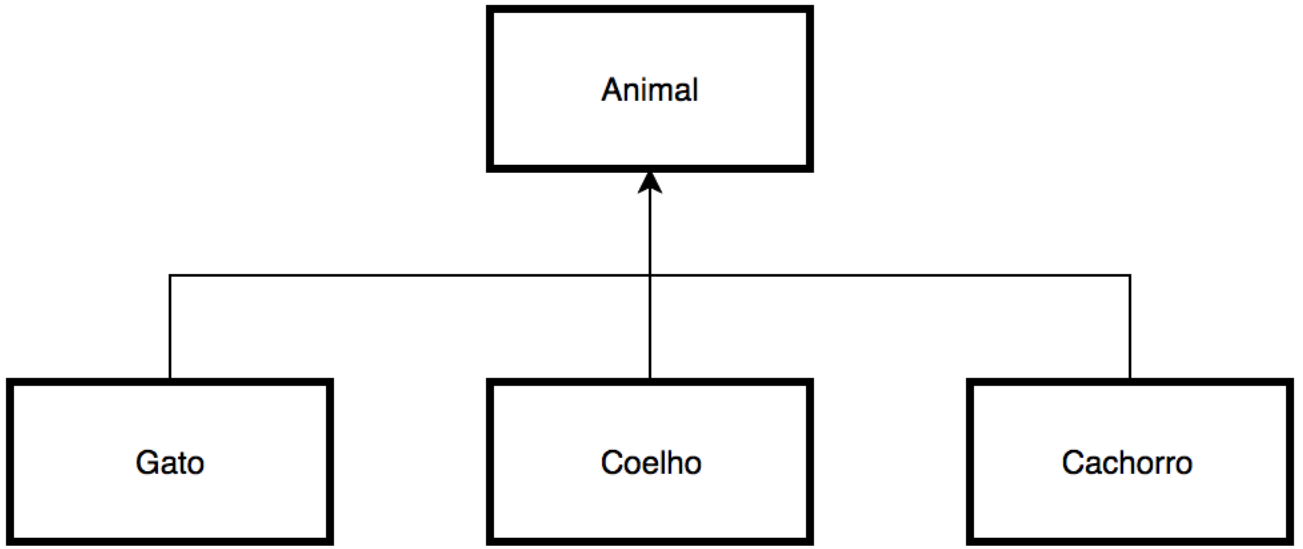
```
class Cachorro:
    def __init__(self,nome,cor):
        self.__nome = nome
        self.__cor = cor

dog = Cachorro(“Bilu”,”Caramelo”)
print(dog.nome)
```

ENCAPSULAMENTO

```python
class Calculadora:
    pass
    def calcular(self,opcao,num1,num2):
        if opcao == '+':
            return self.__adicionar(num1,num2)
        elif opcao == '-':
            return self.__subtrair(num1,num2)
        else:
            print("Opção Inválida!")

    def __adicionar(self,n1,n2):
        return n1 + n2

    def __subtrair(self,n1,n2):
        return n1 + n2

calc = Calculadora()
result = calc.calcular('+',6,10)
print(result)
```

- Também é possível encapsular os métodos.
- Assim somente a classe pode acessar os métodos internamente.

HERANÇA



SUPER CLASSE

```python
class Animal:
    def __init__(self,name,color):
        self._name = name
        self._color = color

    def mover(self):
        print(f" {self._name} ANDOU...")
```

SUB CLASSE

```python
class Peixe(Animal):
    def __init__(self,_name,_color,peso):
        super().__init__(_name,_color)
        self.peso = peso

    def mover(self):
        print(f" {self._name} NADOU...")


if __name__ == "__main__":
    a1 = Animal("BILU","Caramelo")
    a1.mover()

    p1 = Peixe("Nemo","Blue",5)
    p1.mover()
```

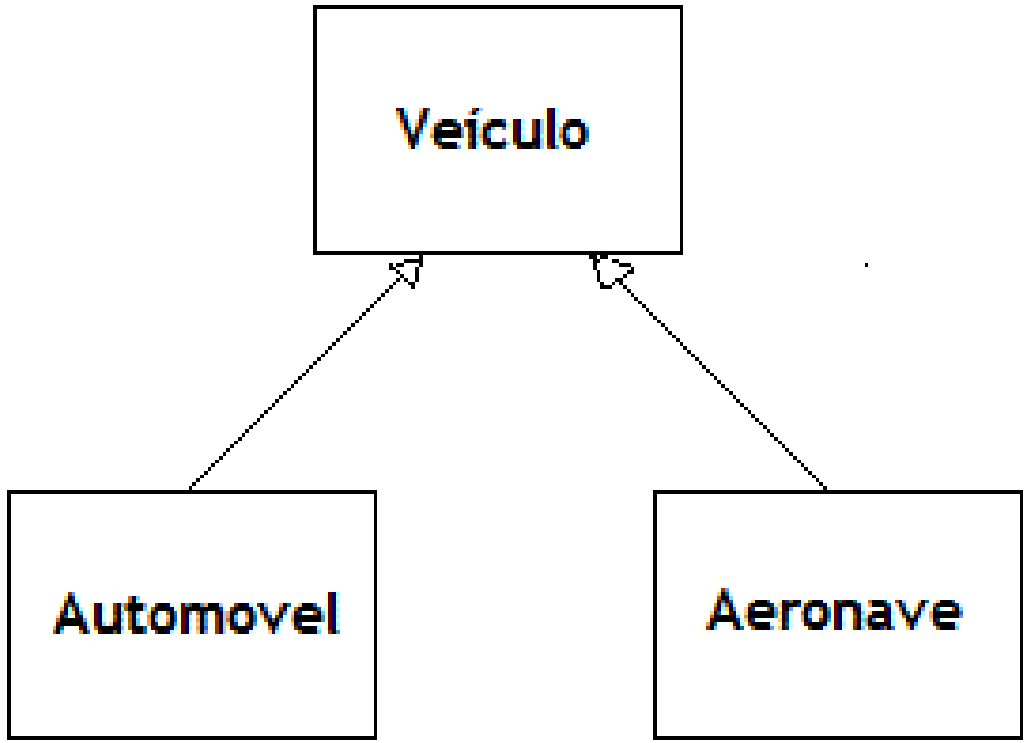
HERANÇA

```
class Peixe(Animal):
    def __init__(self,_name,_color,peso):
        super().__init__(_name,_color)
        self.peso = peso
```

- def __init__(self,_name,_color,peso)
  - herda os atributos da super Classe Animal e recebe um novo argumento para um atributo específico da subclasse nesse caso o peso

- Super(). __init__(_name, _color)
- Invoca o método __init__ da super classe (Animal).

POLIMORFISMO



POLIMORFISMO

- Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C#, PHP, TypeScript, C++ e em Python, que permite ao desenvolvedor usar o mesmo método de formas diferentes. Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem;
- O Polimorfismo acontece na herança, quando a subclasse sobrepõe o método original;
- Você pode redefinir os métodos declarados na super classe de acordo com a especificidade de cada subclasse;

POLIMORFISMO - SUPER CLASSE

```python
class Veiculo:
    def __init__(self, marca=None, modelo=None, ano=None):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano

    def andar(self):
        return (f" {self.modelo} Andou")
```

POLIMORFISMO - SUB CLASSE

```python
class Moto(Veiculo):
    def __init__(self, marca, modelo, ano, motor):
        super().__init__(marca, modelo, ano)
        self.motor = motor

    def andar(self):
        return (f" {self.modelo} Empinou")

motoca = Moto("Honda","FAN",2022,"160s")
print(motoca.andar())
```

EXERCÍCIOS

1 - **Classe Filme**: Crie uma super classe que modele um Filme. Esta classe deve possuir os seguintes atributos:

- Nome;
- Duração;

- **Método:**
  - **Play()**: deve exibir que foi dado play no filme;
- **Subclasses:**
  - **Defina as subclasses de Filme, exemplo Ação, Drama e Suspense. Após a criação das subclasses você deve criar novos métodos específicos a cada subclasse, ex: explodir() em Ação.**

EXERCÍCIOS

2 - **Classe Pessoa**: Crie uma super classe que modele uma Pessoa. Esta classe deve possuir os seguintes atributos:

- Matricula; Nome; Idade;

- **Subclasses:**
  - **Defina as subclasses de Pessoa serão Aluno e Professor, estas devem conter além dos atributos herdados de Pessoa seus atributos identificadores, ex: Classe Aluno (NOTAS; MEDIA).**
  - Classe Professor (Formacao, Disciplina, Carga Horária e Salario)
  - Você deve criar métodos específicos para cada subclasse, ex: calcular_media, estudar, lecionar.

EXERCÍCIOS

3 - **Classe Ingresso**: Crie uma super classe que modele um Ingresso. Esta classe deve possuir os seguintes atributos:

- Preco;
- Setor;

- **Método:**
  - **alterar_preco() e mostrar_setor();**
- **Subclasses:**
  - **Defina a subclasse ingressoVIP com os seguintes atributos: camarote, open_bar, open_food, estacionamento -> todos booleanos, True ou False;**
  - Acrescente os métodos pegar_bebida() e acessar_camarote();

EXERCÍCIOS

4 - **Classe Passagem**: Crie uma super classe que modele uma Passagem. Esta classe deve possuir os seguintes atributos:

- Preco;
- Assento;

- **Método:**
  - **alterar_preco() e escolher_assento();**
- **Subclasses:**
  - **Defina a subclasse PassagemBus e PassagemAviao com os seguintes atributos: portaodeembarque e checkin para classe PassagemAviao, placa e leito par PassagemBus;**
  - Crie um novo método específico para cada subclasse. Ex: decolar() e abastecer()

EXERCÍCIOS

5 - **Classe Pessoa**: Crie uma super classe que modele uma Pessoa. Esta classe deve possuir os seguintes atributos:

- Nome; Telefone; E-mail; Endereço;

- **Métodos:**
  - **negociar**: deve printar uma mensagem de negociação;
- **Subclasses:**
  - **Defina as subclasses de Pessoa serão Física e Jurídica, estas devem conter além dos atributos herdados de Pessoa seus atributos identificadores, ex: CPF, CNPJ.**
  - Além de herdar o método negociar() crie métodos específicos para as subclasses;

EXERCÍCIOS

6 - **Classe Funcionário**: Crie uma super classe que modele um Funcionário genérico. Esta classe deve possuir os seguintes atributos:

- Nome;
- Matricula;
- Salario;

- **Método:**
  - **Bater_ponto()**: deve criar uma lista de pontos do funcionário, pode ser booleana 0 ou 1;
- **Subclasses:**
  - **Defina as subclasses de Funcionário, exemplo Vendedor e Gerente. Após a criação das subclasses você deve criar atributos e métodos específicos de cada subclasse;**
  - Ex: atributo comissão e método bater_meta() para Vendedor e atributo senha para o Gerente.

EXERCÍCIOS

7 - **Classe Brinquedos**: Andy Davis precisa classificar seus brinquedos por Subclasses, sabemos que cada brinquedo tem atributos e métodos diferentes, exemplo Buzz Lightyear voa e Woody laça. Defina principais atributos:

- Nome, Cor, Tamanho, Preço;

- **Método:**
  - brincar(); - fazer um print simples, estou brincando com {nome do brinquedo}
- **Subclasses:**
  - Crie 10 sub classes de brinquedos com seus respectivos atributos e métodos.
  - Utilize o polimorfismo para reescrever o método herdado da super classe

EXERCÍCIOS

8 - **Classe Imóvel**: Uma Imobiliária precisa de um sistema que controle o aluguel de seus Imóveis. Para isto você deve definir em um módulo a super classe Imóvel com os seguintes atributos:

- InscricaoMunicipal; Valor_aluguel; IPTU;

- **Método:**
  - **obter_parcela_IPTU();**
  - **Set_valor_aluguel();**
- **Subclasses:**
  - **Defina as subclasses de Imóvel sendo: Casa, Condomínio; Apartamento; Terreno e Chácara;**
  - Defina os atributos específicos para cada sub classe, exemplo: piscina, sala_de_estar,
  - Quartos, churrasqueira, área m², elevador, área_de_lazer, .

EXERCÍCIOS

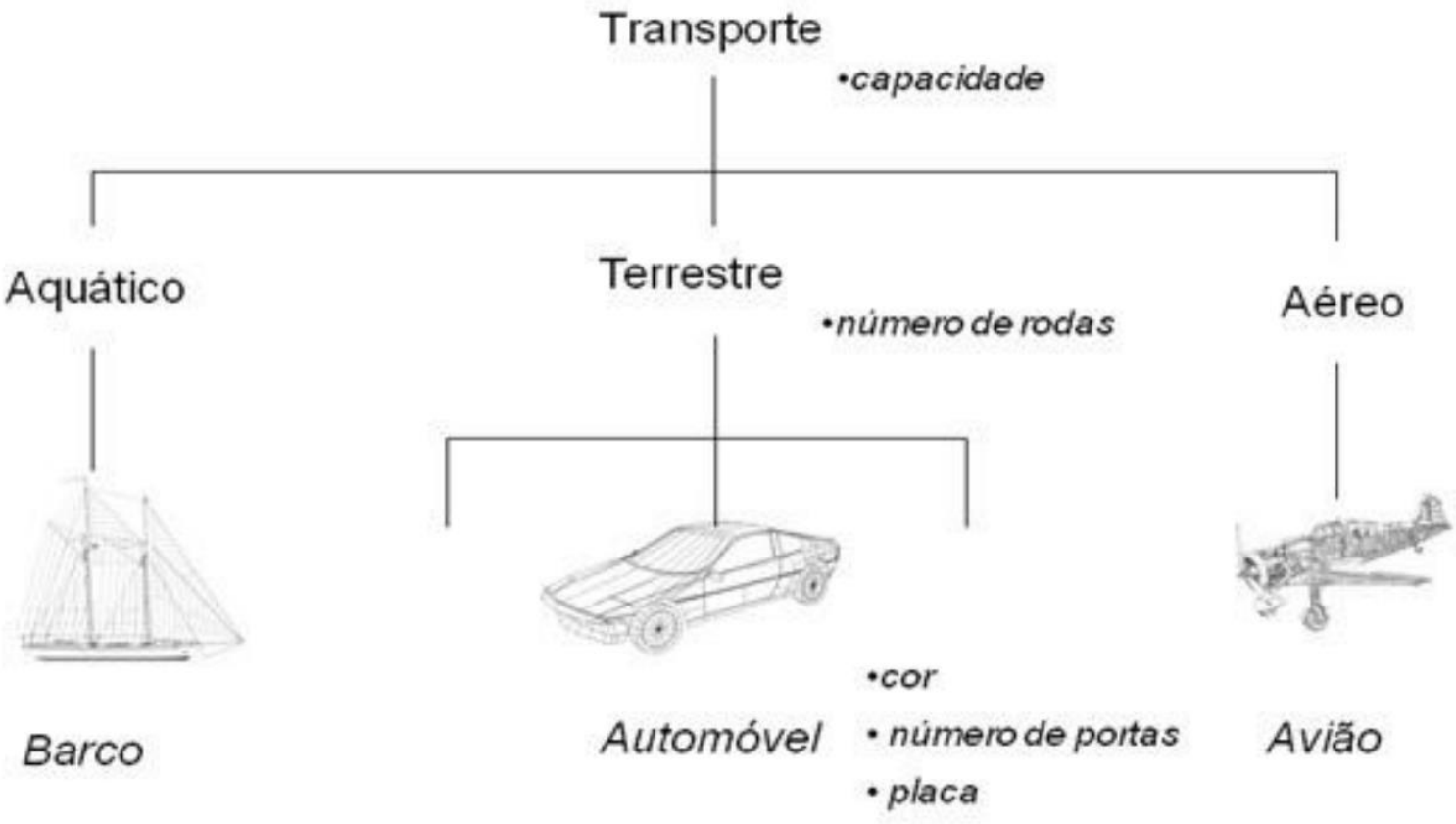
9 - **Classe Compra**: Crie uma super classe que modele uma Venda. Esta classe deve possuir os seguintes atributos:

- Numero; Produto; Valor; Valor_total = 0;

- **Método:**
  - **calcular_valor_total()**: deve somar ao valor_total o imposto de 17% do ICMS + o Frete de 5% sobre o valor do produto;

- **Subclasses:**
  - **Defina as subclasses Avista e Parcelada, na classe de compra a vista deve ter o atributo desconto e na classe Parcelada numero de parcelas.**
  - Em cada subclasse definir um método que retorna o preço com desconto ou o valor das parcelas.

10 - **Classe Transporte**: Crie uma super classe Transporte e suas respectivas subclasses, sendo Terrestre e uma terceira subclasse de transporte Automóvel, modele tipos de transportes de acordo com a imagem abaixo:



EXERCÍCIOS

10 - **Classe Transporte**: Você deve analisar a hierarquia do transporte tipo Terrestre e da Classe automóvel para criar as subclasses dos tipos de transporte Aéreo e Aquático. Instancie 3 objetos de cada classe e faça os testes nos atributos e métodos específicos de cada subclasse;

- Crie duas subclasses de Aquático e Aéreo, exemplo:
  - Lancha e Navio;
  - AviaoMonomotor e AviaoComercial;
- Verifique os atributos e métodos específicos de cada subclasse.