youth! space

PROJETO </> EDUCAÇÃO

DO FUTURO

MÓDULO 03

POO

Programação Orientada a Objetos

AULA 2

AULA 2

CONTEÚDOS

- Orientação à Objeto
- Abstração
- Encapsulamento
- Herança
- Função Super
- Polimorfismo

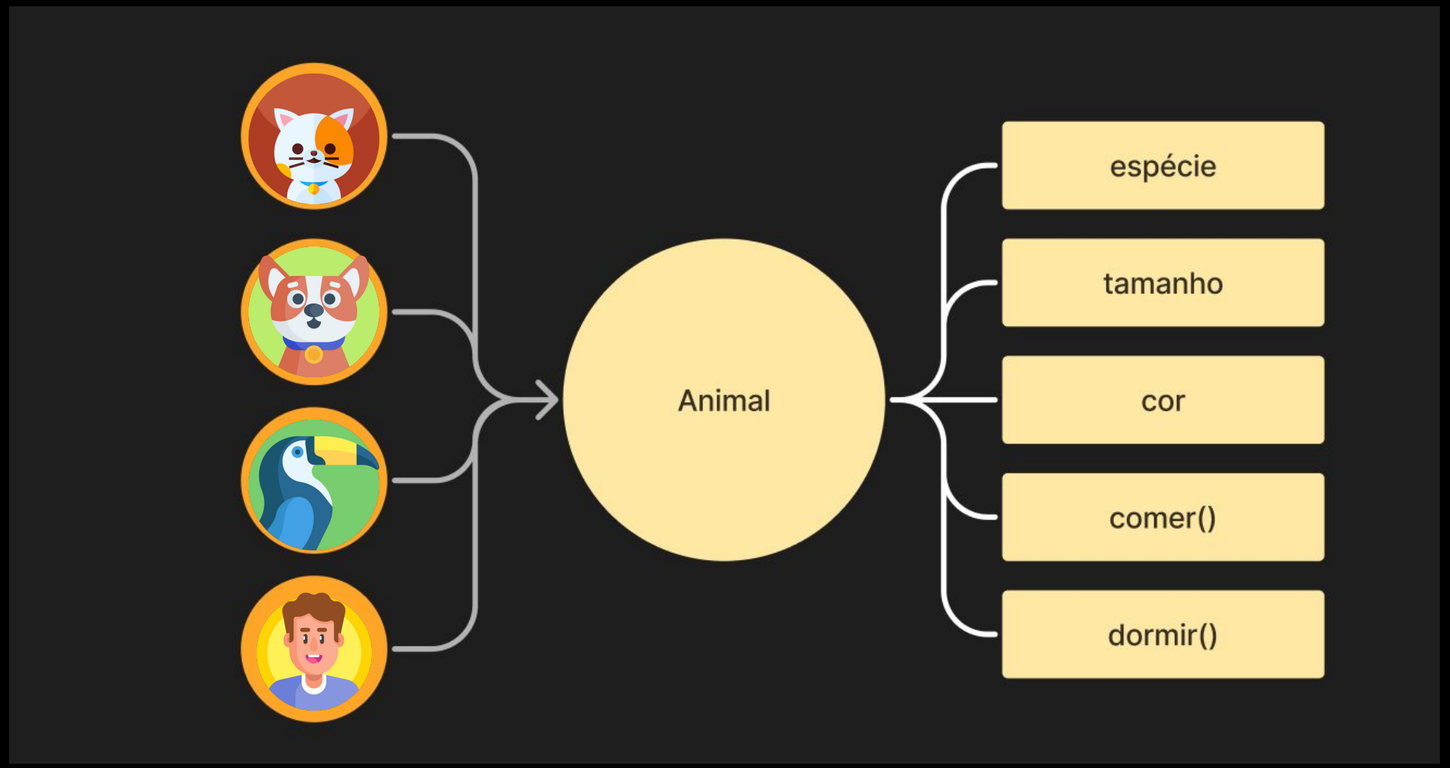
Orientação a objeto

A programação orientada a objetos tem como objetivo abstrair elementos reais, tangíveis ou não, por meio de classes, no qual cada uma dessas classes possui propriedades únicas e comportamentos próprios, podendo existir só ou dependendo de outra.

Um dos principais objetivos da POO é facilitar o entendimento e a modelagem de software complexo através da organização do código em unidades lógicas que representam entidades ou conceitos do mundo real. Para isso lançamos mão dos 4 pilares da Orientação a Objeto

ABSTRAÇÃO

Abstração é o conceito de simplificar complexidades desnecessárias, focando nos aspectos relevantes de um objeto.
Em POO, uma classe é uma forma de abstração, pois representa um conjunto de objetos que compartilham características semelhantes. Por exemplo, ao criar uma classe Animal, estamos abstraindo os detalhes específicos de cada tipo de animal e focando apenas nos aspectos comuns a todos os animais.



O conceito de abstração se desdobra em:

Classes:
São conjuntos de objetos que possuem o mesmo tipo; e

Objetos:
São instâncias das classes.

ABSTRAÇÃO

Abstração é o conceito de simplificar complexidades desnecessárias, focando nos aspectos relevantes de um objeto.
Em POO, uma classe é uma forma de abstração, pois representa um conjunto de objetos que compartilham características semelhantes. Por exemplo, ao criar uma classe Animal, estamos abstraindo os detalhes específicos de cada tipo de animal e focando apenas nos aspectos comuns a todos os animais.

```
class Animal:
    def   init  (self, raca, tamanho, cor):
        self.raca = raca
        self.tamanho = tamanho
        self.cor = cor

    def comer(self):
        return "O animal está comendo."

    def dormir(self):
        return "O animal está dormindo."
```

O conceito de abstração se desdobra em:

Classes:
São conjuntos de objetos que possuem o mesmo tipo; e

Objetos:
São instâncias das classes.

ATIVIDADE 1

Você foi contratado para desenvolver um sistema de gerenciamento de biblioteca para uma instituição de ensino.

O sistema deve permitir que os usuários realizem operações como:
Criar um novo livro

- Título
- Autor
- Ano de Publicação
- Status (disponível ou emprestado)

Criar uma biblioteca para reunir os livros.
Os objetos do tipo biblioteca contarão com as seguintes funções:

- adicionar livros à biblioteca
- listar os livros disponíveis.

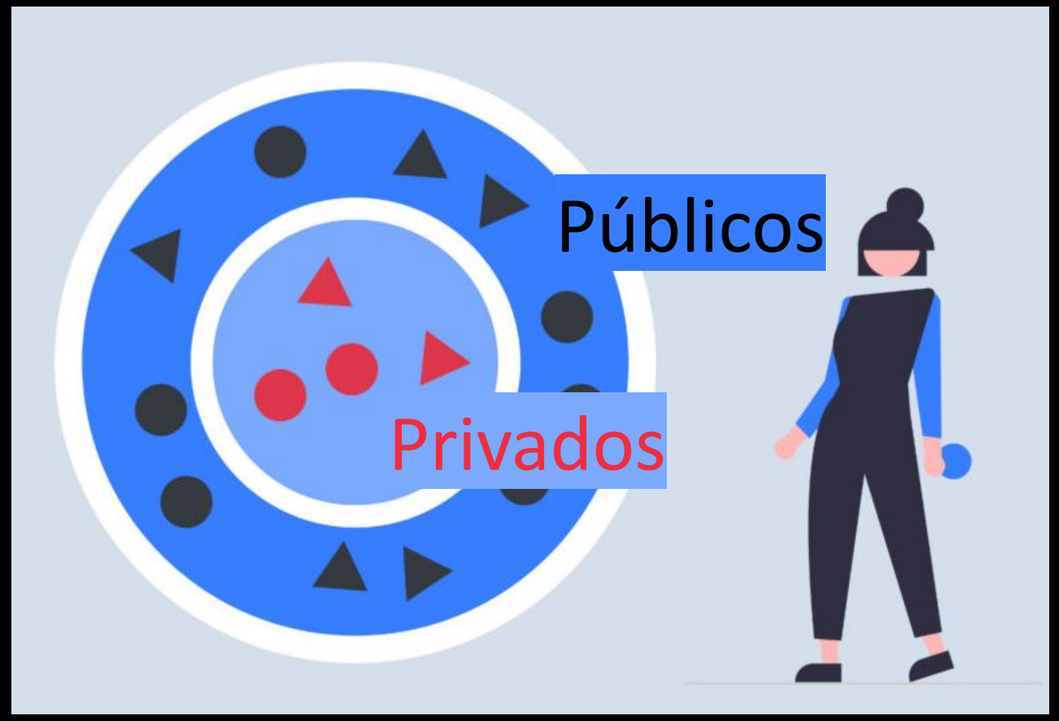
ENCAPSULAMENTO

O encapsulamento é a habilidade de esconder as características intrínsecas de um dado objeto de outros objetos. É a técnica de proteger os dados (atributos) e os códigos que operam sobre os dados (métodos) em uma única unidade chamada classe. O encapsulamento protege os dados de um objeto contra acesso direto de fora da classe, expondo apenas métodos seguros para operação sobre esses dados.

O conceito de encapsulamento se desdobra em:

Métodos ou atributos públicos: são aqueles que podem ser acessados de fora da classe

Métodos ou atributos privados: são aqueles que devem ser acessados apenas de dentro da classe.



ENCAPSULAMENTO

O conceito de encapsulamento se desdobra em:

Métodos ou atributos públicos: são aqueles que podem ser acessados de fora da classe

Métodos ou atributos privados: são aqueles que devem ser acessados apenas de dentro da classe.

```python
class Pessoa:
    def __init__(self, nome, idade,cpf):
    self._nome = nome  # Atributo protegido
    self._idade = idade # Atributo protegido
    self.__cpf = cpf # Atributo privado

    def cumprimentar(self):  # Método público
    return f"Olá, meu nome é {self._nome}."

    def __aniversario(self):  # Método privado
    return f"Parabéns pelo seu aniversário, {self._nome}!"

    def envelhecer(self, anos):  # Método público
    self._idade += ano
    self.__aniversario()
    return f"Agora tenho {self._idade} anos."



pessoa = Pessoa("Ana", 30)
print(pessoa.cumprimentar())  # Acesso protegido, acessado através de um método público
print(pessoa.envelhecer(5))  # Acesso protegido, acessado através de um método público
print(pessoa.__aniversario())  # Acesso privado, não permitido
print(pessoa.__cpf) # Acesso privado, não permitido
```

ATIVIDADE 2

Iremos evoluir o sistema da nossa biblioteca. Agora além de listar os livros, a Classe Biblioteca deverá contar com duas opções de métodos:

- emprestarLivro
- devolverLivro.

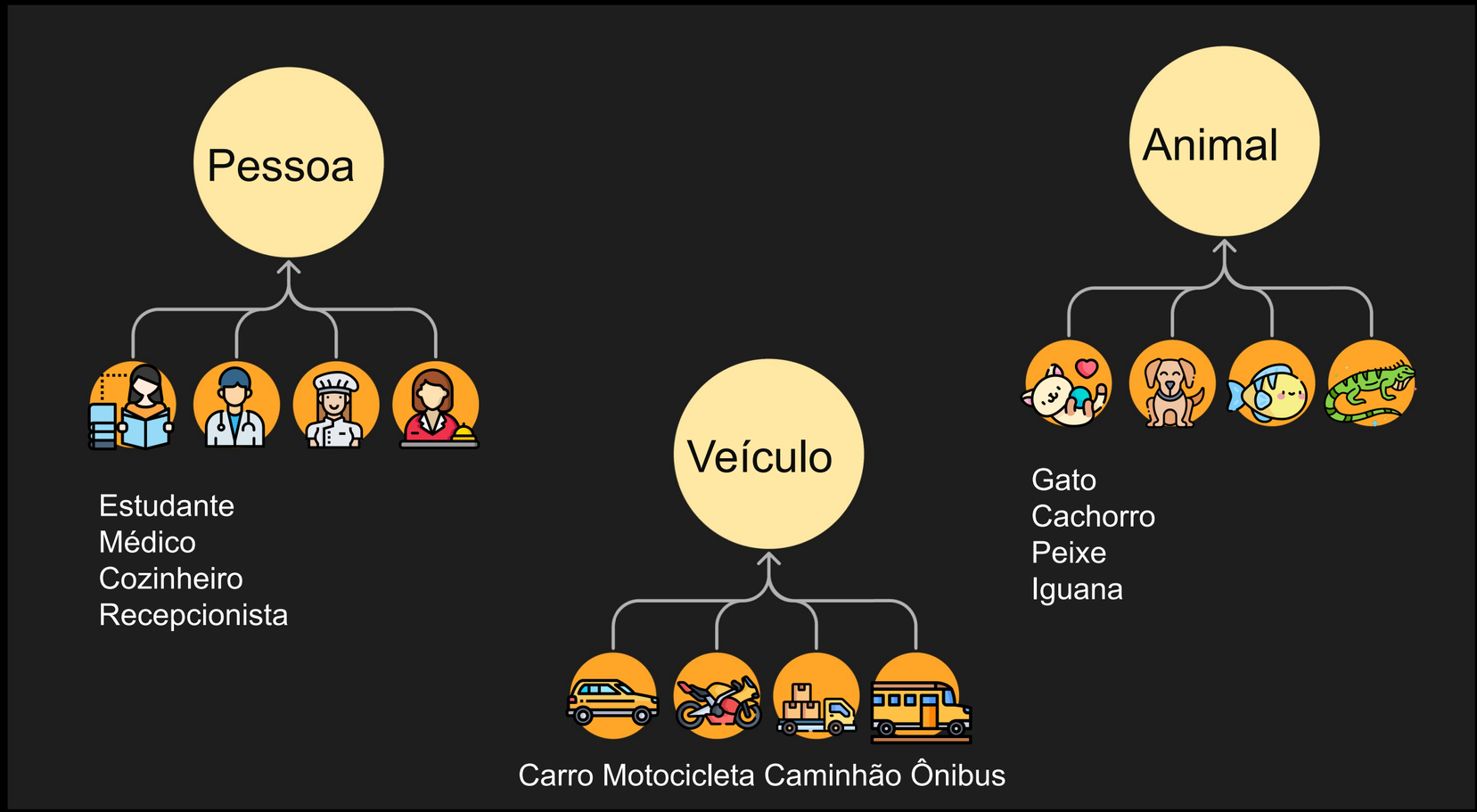
Dessa vez o atributo status deverá ser colocado como privado para Classe Livro . Por sua vez ela deverá contar com dois métodos para fazer a alteração do status, já que eles serão um atributo privado. Escolha o melhor nome para os métodos.

HERANÇA

A **herança** é um mecanismo pelo qual uma nova classe, chamada classe derivada ou subclasse, pode adquirir as propriedades de uma classe existente, conhecida como classe base ou superclasse. Esse pilar permite a reutilização de código, a extensibilidade e a organização hierárquica de classes

O conceito de herança se desdobra em:

A herança promove a ideia de generalização para especialização, onde classes mais genéricas são estendidas por classes mais específicas, adicionando novos atributos ou comportamentos ou modificando os existentes.



HERANÇA

A **herança** é um mecanismo pelo qual uma nova classe, chamada classe derivada ou subclasse, pode adquirir as propriedades de uma classe existente, conhecida como classe base ou superclasse. Esse pilar permite a reutilização de código, a extensibilidade e a organização hierárquica de classes

O conceito de herança se desdobra em:

A herança promove a ideia de generalização para especialização, onde classes mais genéricas são estendidas por classes mais específicas, adicionando novos atributos ou comportamentos ou modificando os existentes.

```python
class Pessoa:
    def __init__(self, nome, idade):
        self._nome = nome
        self._idade = idade

    def cumprimentar(self):
        return f"Olá, meu nome é {self._nome}."

class Estudante(Pessoa):
    def __init__(self, nome, idade, curso):
        super().__init__(nome, idade)
        self._curso = curso

    def estudar(self):
        return f"{self._nome} está estudando
{self._curso}."
```

```python
# Exemplo de uso:
pessoa = Pessoa("João", 25)
print(pessoa.cumprimentar())

estudante = Estudante("Maria",
20, "Engenharia")
print(estudante.cumprimentar())
print(estudante.estudar())
```

HERANÇA - super ()

Super() em Python é uma função embutida que fornece uma maneira de acessar métodos e atributos da classe pai (ou superclasse) durante a herança. Ele é comumente usado dentro de métodos de uma classe derivada (ou subclasse) para chamar métodos da classe pai.

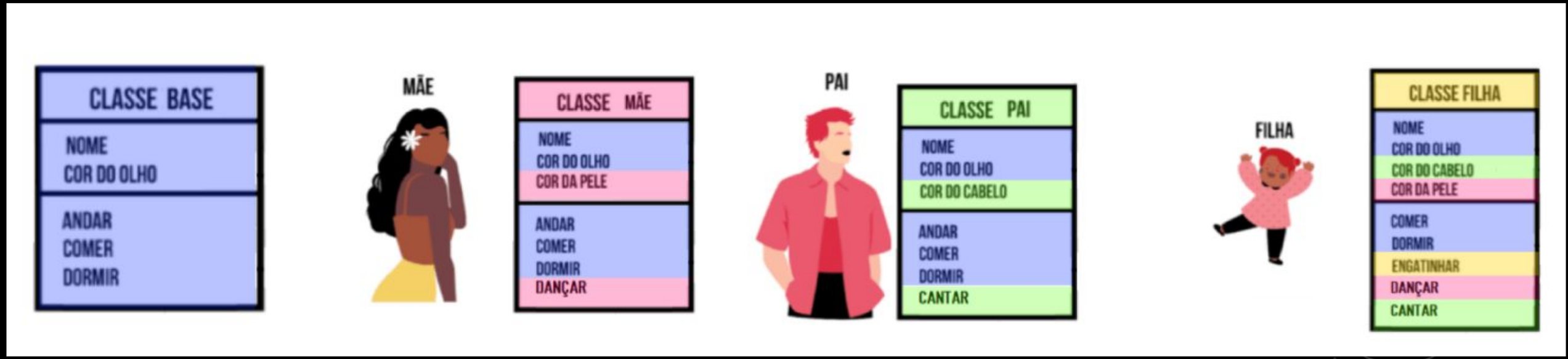
```
class Pessoa:
    def __init__(self, nome, idade):
        self._nome = nome
        self._idade = idade

    def cumprimentar(self):
        return f"Olá, meu nome é {self._nome}."
```

```
class Estudante(Pessoa):
    def __init__(self, nome, idade, curso):
        super().__init__(nome, idade)
        self._curso = curso

    def estudar(self):
        return f"{self._nome} está estudando
{self._curso}."
```

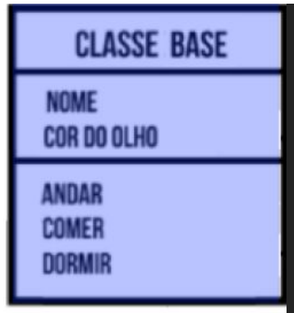
```
pessoa = Pessoa("João", 25)
print(pessoa.cumprimentar())
estudante = Estudante("Maria", 20, "Engenharia")
print(estudante.cumprimentar())
print(estudante.estudar())
```

HERANÇA - MÚLTIPLA

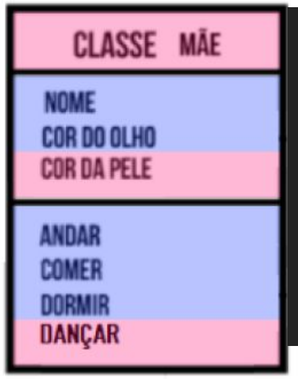
A herança múltipla é um recurso da programação orientada a objetos que permite a uma classe herdar atributos e métodos de mais de uma classe base. Permite reutilizar o código de múltiplas classes base, promovendo a modularidade e a manutenção do código. Facilita a criação de classes que combinam funcionalidades diversas, sem a necessidade de reescrever o código existente.



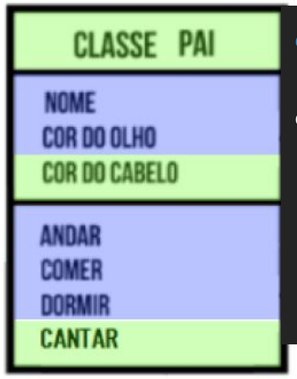
HERANÇA - MÚLTIPLA




```python
class Base:
    def __init__(self, nome, cor_olho):
        self.nome = nome
        self.cor_olho = cor_olho

    def andar(self):
        return f"{self.nome} está andando."

    def comer(self):
        return f"{self.nome} está comendo."

    def dormir(self):
        return f"{self.nome} está dormindo."
```

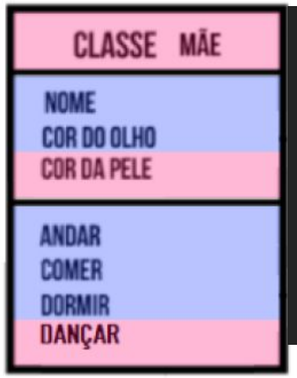



```python
class Mae(Base):
    def __init__(self, nome, cor_olho, cor_pele):
        super().__init__(nome, cor_olho)
        self.cor_pele = cor_pele

    def dancar(self):
        return f"{self.nome} está dançando."
```

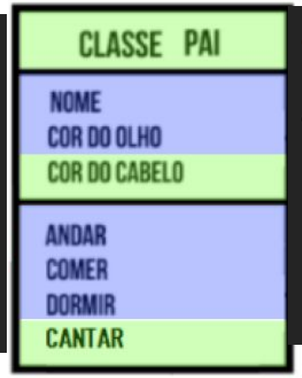



```python
class Pai(Base):
    def __init__(self, nome, cor_olho, cor_cabelo):
        super().__init__(nome, cor_olho)
        self.cor_cabelo = cor_cabelo

    def cantar(self):
        return f"{self.nome} está cantando."
```

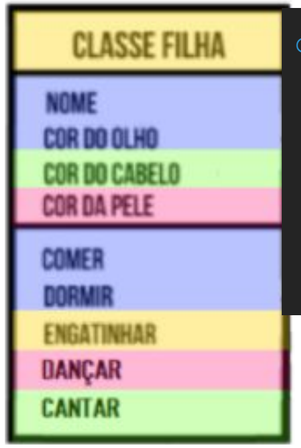
HERANÇA- MÚLTIPLA




```python
class Mae(Base):
    def __init__(self, nome, cor_olho,
cor_pele):
        super().__init__(nome, cor_olho)
        self.cor_pele = cor_pele

    def dancar(self):
        return f"{self.nome} está dançando."
```




```python
class Pai(Base):
    def __init__(self, nome, cor_olho,
cor_cabelo):
        super().__init__(nome, cor_olho)
        self.cor_cabelo = cor_cabelo

    def cantar(self):
        return f"{self.nome} está cantando."
```




```python
class Filha(Mae, Pai):
    def __init__(self, nome, cor_olho, cor_pele, cor_cabelo):
        Mae.__init__(self, nome, cor_olho, cor_pele)
        Pai.__init__(self, nome, cor_olho, cor_cabelo)

    def engatinhar(self):
        return f"{self.nome} está engatinhando."
```

ATIVIDADE 3

Você foi contratado para desenvolver um sistema de gerenciamento de veículos para uma concessionária de automóveis. O sistema deve permitir que a concessionária gerencie diferentes tipos de veículos, como carros, motos e caminhões.

Classe base - Veiculo:

- Marca
- Modelo
- Ano de Fabricação
- Preço
- Exibir informações

Classe derivada - Carro:

- número de portas
- tipo de combustível

Classe derivada - Moto:

- cilindrada
- tipo de partida

Classe - Concessionaria (armazena uma lista de veículos)

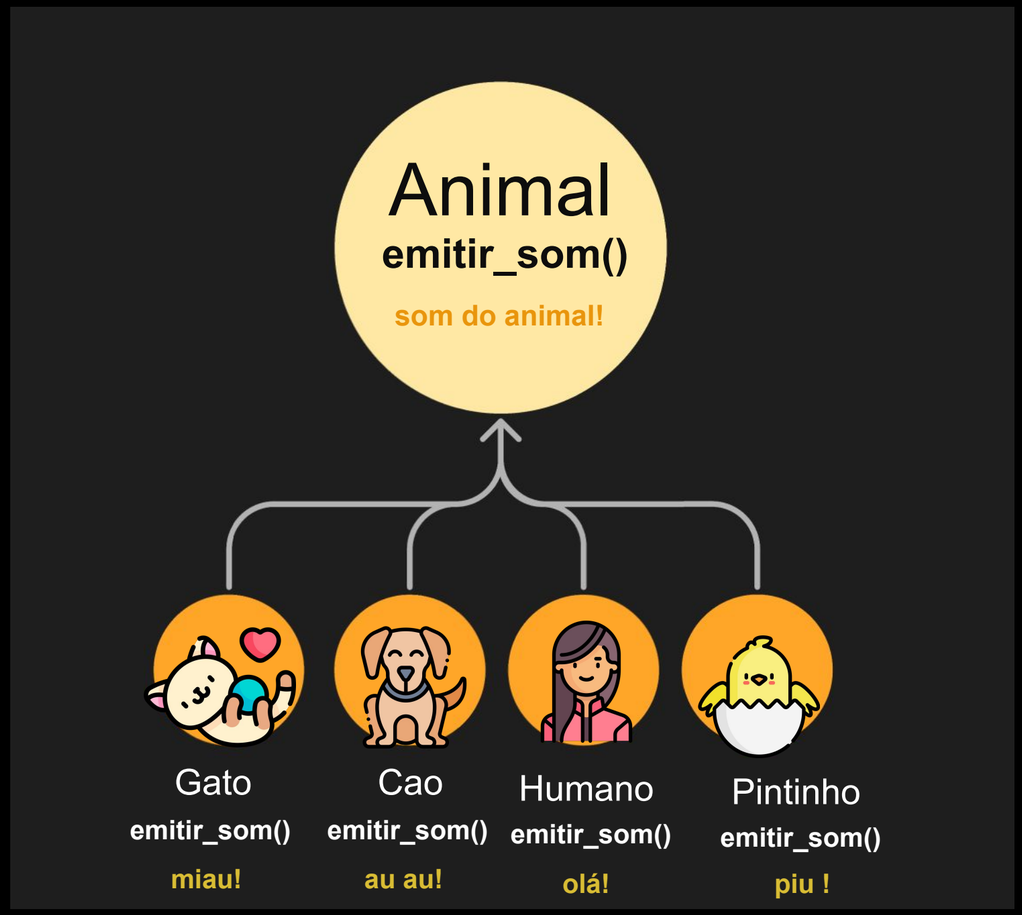
- Adicionar um veículo à lista.
- Listar todos os veículos disponíveis na concessionária.

POLIMORFISMO

O polimorfismo é um dos princípios fundamentais da programação orientada a objetos e refere-se à capacidade de diferentes objetos responderem ao mesmo método de maneiras diferentes.

O conceito de polimorfismo se desdobra em:

Classes derivadas podem ter implementações específicas para métodos definidos em uma classe base, de modo que esses métodos possam ser chamados de maneira uniforme, independentemente do tipo específico do objeto.



POLIMORFISMO

O polimorfismo é um dos princípios fundamentais da programação orientada a objetos e refere-se à capacidade de diferentes objetos responderem ao mesmo método de maneiras diferentes.

O conceito de polimorfismo se desdobra em:

Classes derivadas podem ter implementações específicas para métodos definidos em uma classe base, de modo que esses métodos possam ser chamados de maneira uniforme, independentemente do tipo específico do objeto.

```
class Animal:
    def emitir_som(self):
       print ( "som genérico de um animal!")

class Gato(Animal):
    def emitir_som(self):
        print ( " Miau!" )

class Humano(Animal):
    def emitir_som(self):
        print ( "Olá, tudo bem?")


gatinho = Gato()
pessoa = Humano()

gatinho.emitir_som() # Miau!
pessoa.emitir_som()  # Olá, tudo bem?!
```

ATIVIDADE 4

As classes Carros e Moto devem fazer a modificação da função Exibir informações da superclasse Veículos de forma a mostrar os atributos específicos delas.

DESAFIO

Você foi contratado para desenvolver um sistema de supermercado que aborde os 4 pilares principais da Programação Orientada a Objetos (POO): abstração, encapsulamento, herança e polimorfismo.

O sistema deverá gerenciar um estoque de produtos:

- alimentos
- bebidas

Deverá ainda e permitir operações :

- adicionar
- remover
- listar produtos

DESAFIO - REQUISITOS DO PROJETO

- Encapsulamento:
  atributos das classes devem ser acessíveis apenas por meio de métodos

- Herança:
  A classe base Produto define os atributos comuns a todos os produtos. As subclasses Alimento e Bebida que herdam da classe Produto.

- Polimorfismo:
  O métodos descrição serão sobrescritos, para fornecer informações específicas sobre o produto.

Funcionalidades Esperadas:

1. Adicionar um novo produto ao estoque.
2. Remover um produto do estoque.
3. Listar todos os produtos disponíveis no estoque, mostrando suas informações.
4. Mostrar produtos indisponíveis.

(85) 98524-9935 youthidiomas

contato@youthidiomas.com.br

www.youthidiomas.com.br

2023