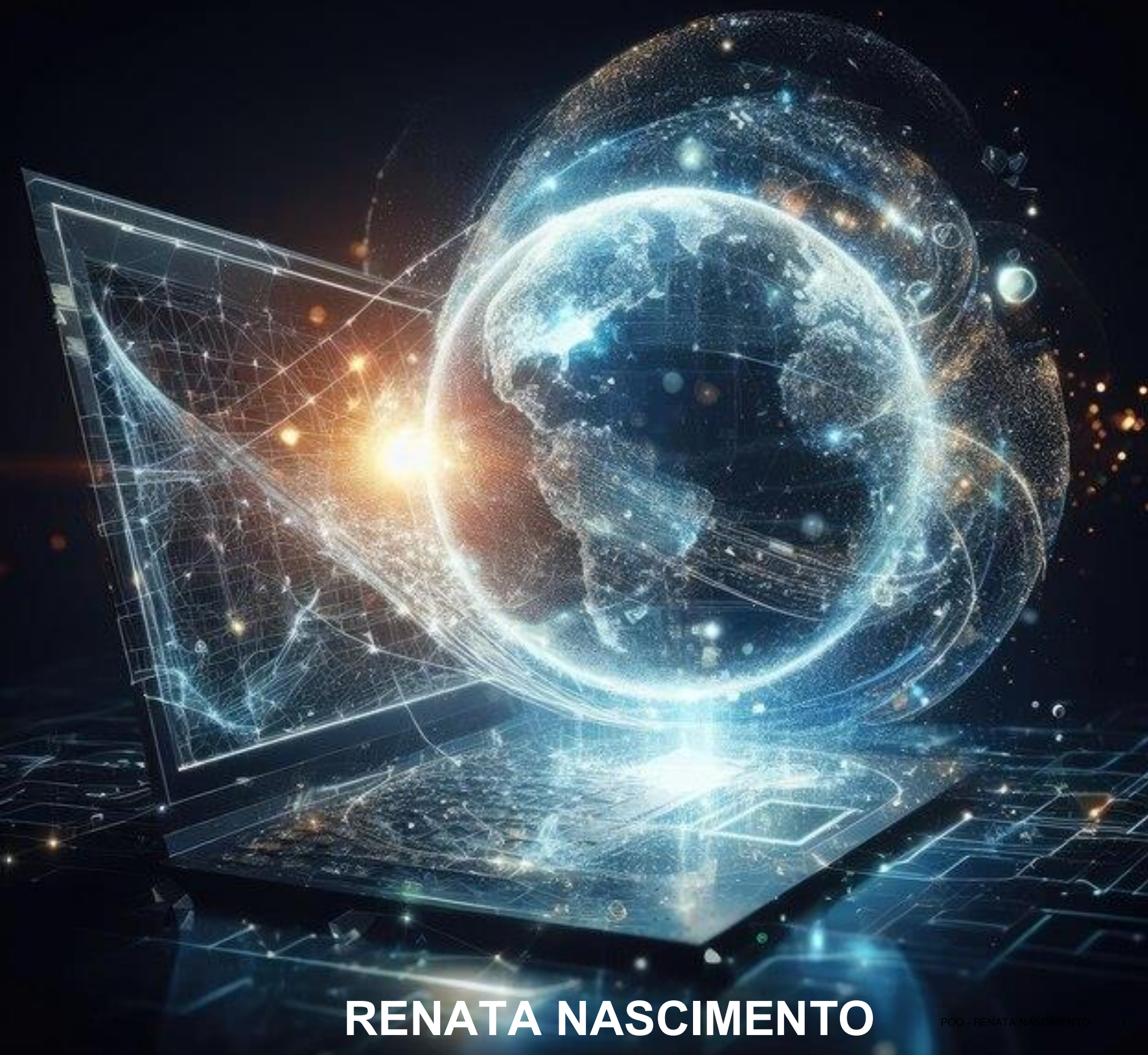
POO SEM MISTÉRIOS:
A JORNADA DO INICIANTE



RENATA NASCIMENTO

Introdução

A Programação Orientada a Objetos (POO) é um paradigma amplamente utilizado que organiza o código em torno de objetos que representam elementos do mundo real. Vamos explorar seus quatro pilares principais: encapsulamento, herança, polimorfismo e abstração, com exemplos simples e práticos.

01
ENCAPSOLAMENTO

1. Encapsulamento: Protegendo os Dados

Encapsulamento significa esconder os detalhes internos de uma classe e permitir o acesso aos dados apenas por meio de métodos específicos. Isso garante segurança e organização no código.

Exemplo prático: Conta Bancária

```python
class ContaBancaria:
    def __init__(self, titular, saldo_inicial):
        self.titular = titular
        self.__saldo = saldo_inicial  # Variável privada

    def depositar(self, valor):
        if valor > 0:
            self.__saldo += valor
            print(f"Depósito de R${valor} realizado com sucesso!")

    def sacar(self, valor):
        if 0 < valor ⩽ self.__saldo:
            self.__saldo -= valor
            print(f"Saque de R${valor} realizado com sucesso!")
        else:
            print("Saldo insuficiente ou valor inválido.")

    def exibir_saldo(self):
        print(f"Saldo atual: R${self.__saldo}")

# Uso
conta = ContaBancaria("João", 1000)
conta.depositar(500)
conta.sacar(300)
conta.exibir_saldo()
```

02
HERANÇA

2. Herança: Reaproveitando Código

Herança permite criar uma nova classe que reutiliza e especializa o comportamento de uma classe existente.

Exemplo prático: Veículos

```python
class Veiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def exibir_info(self):
        print(f"Marca: {self.marca}, Modelo: {self.modelo}")

class Carro(Veiculo):
    def __init__(self, marca, modelo, portas):
        super().__init__(marca, modelo)
        self.portas = portas

    def exibir_info(self):
        super().exibir_info()
        print(f"Portas: {self.portas}")

# Uso
meu_carro = Carro("Toyota", "Corolla", 4)
meu_carro.exibir_info()
```

03
POLIMORFISMO

3. Polimorfismo: Comportamento Diferente para o Mesmo Método

Polimorfismo é a capacidade de um mesmo método ter comportamentos diferentes dependendo do contexto.

Exemplo prático: Pagamento

```python
class Pagamento:
    def realizar_pagamento(self, valor):
        pass

class PagamentoCartao(Pagamento):
    def realizar_pagamento(self, valor):
        print(f"Pagamento de R${valor} realizado com cartão.")

class PagamentoPix(Pagamento):
    def realizar_pagamento(self, valor):
        print(f"Pagamento de R${valor} realizado via PIX.")

# Uso
formas_pagamento = [PagamentoCartao(), PagamentoPix()]
for forma in formas_pagamento:
    forma.realizar_pagamento(100)
```

04
ABSTRAÇÃO

4. Abstração: Focando no Essencial

Abstração significa destacar os aspectos mais relevantes de um objeto, escondendo detalhes desnecessários para o usuário final.

Exemplo prático: Sistema de Pedidos

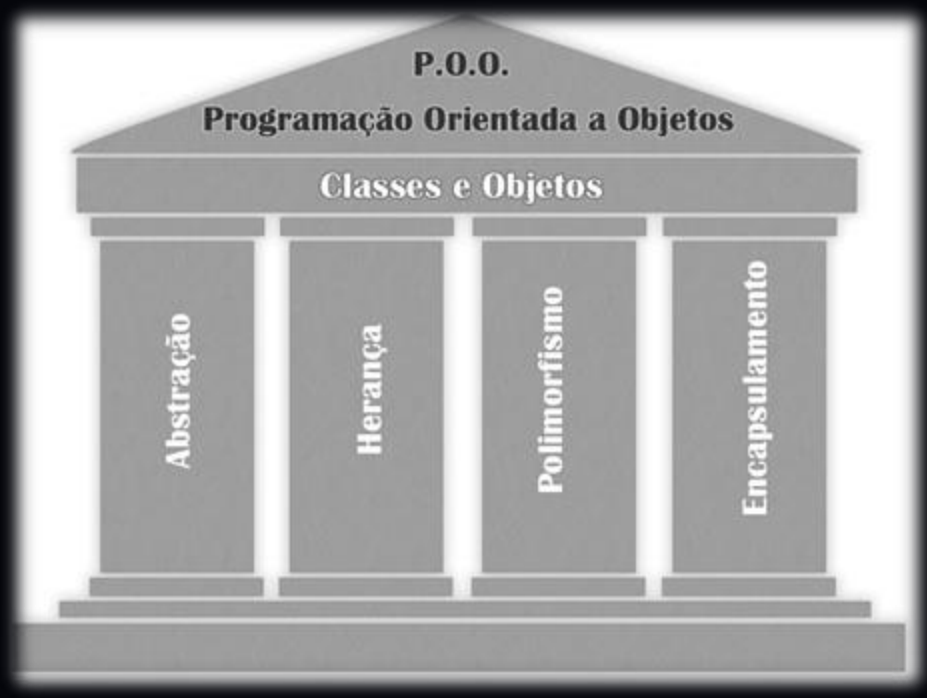
```python
from abc import ABC, abstractmethod

class Pedido(ABC):
    @abstractmethod
    def calcular_total(self):
        pass

class PedidoOnline(Pedido):
    def __init__(self, itens, taxa_entrega):
        self.itens = itens
        self.taxa_entrega = taxa_entrega

    def calcular_total(self):
        return sum(self.itens) + self.taxa_entrega

# Uso
pedido = PedidoOnline([50, 100, 30], 20)
print(f"Total do pedido: R${pedido.calcular_total()}")
```



Os pilares da POO ajudam a criar códigos organizados, reutilizáveis e seguros. Ao compreender e aplicar encapsulamento, herança, polimorfismo e abstração, você estará no caminho para se tornar um programador mais eficiente e preparado para desafios reais.

Agradecimentos

Obrigada por ler este ebook.

Esse ebook foi gerado por IA e diagramado por humano. Seu passo a passo encontra-se no github.

Esse ebook foi gerado para fins didáticos e não foi totalmente realizada validação humana

https://github.com/Renatagenne28