

Abstração, Herança e Polimorfismo

Esta apresentação vai explorar três pilares fundamentais da programação orientada a objetos (POO): abstração, herança e polimorfismo.

Matheus da Silva Xavier

Keven Iuri Souza dos Santos

Santiago Ramos de Lima



por keven iuri

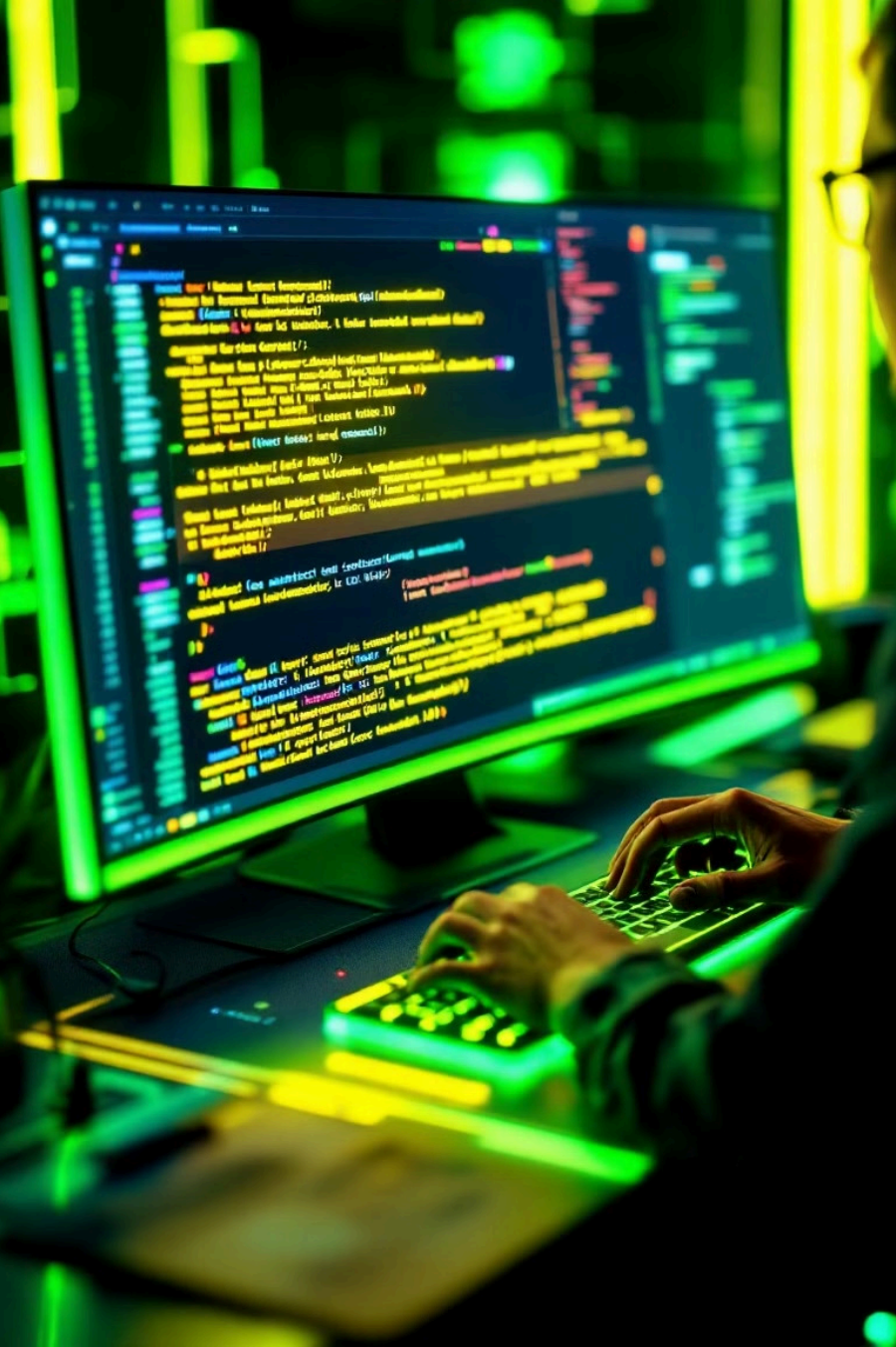
Abstração: Simplificando a Complexidade

O Essencial, Sem Detalhes

Abstração significa esconder detalhes de implementação e expor apenas o essencial. O foco está no "o que" a funcionalidade faz, e não no "como" ela funciona.

O Poder da Simplificação

Imagine um carro. Você sabe dirigir (usar a interface), mas não precisa saber como o motor funciona internamente. A abstração permite que você se concentre na tarefa em mãos, sem se preocupar com detalhes complexos.



Objetivo

1

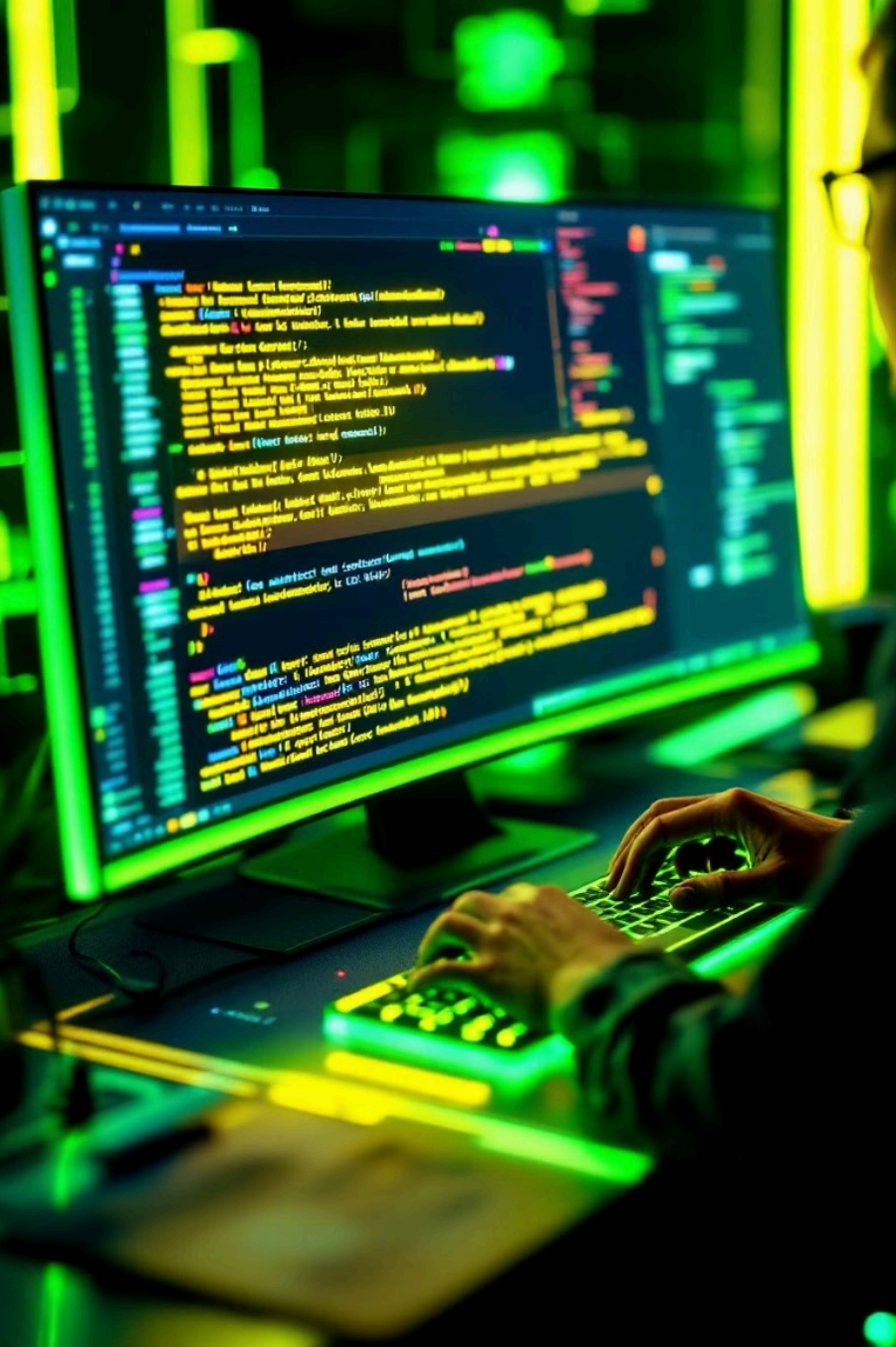
Facilitar o entendimento

2

Reutilização

3

Manutenção do código



Por que Utilizamos Abstração?

1

Simplicidade: Reduz a complexidade do sistema, deixando o código mais legível.

2

Reusabilidade: Podemos usar componentes de forma modular em várias partes do código

3

Facilidade de Manutenção: Mudanças nos detalhes de implementação não afetam o restante do sistema.


```
from abc import ABC, abstractmethod
```

```
class FormaGeometrica(ABC):
```

```
@abstractmethod
def calcular_area(self):
    pass

@abstractmethod
def calcular_perimetro(self):
    pass
```

```
class Retangulo(FormaGeometrica):
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura
```

```
    def calcular_area(self):
        return self.largura * self.altura

    def calcular_perimetro(self):
        return 2 * (self.largura + self.altura)
```

```
class Circulo(FormaGeometrica):
    def __init__(self, raio):
        self.raio = raio
```

```
    def calcular_area(self):
        return 3.1415 * (self.raio ** 2)

    def calcular_perimetro(self):
        return 2 * 3.1415 * self.raio
```

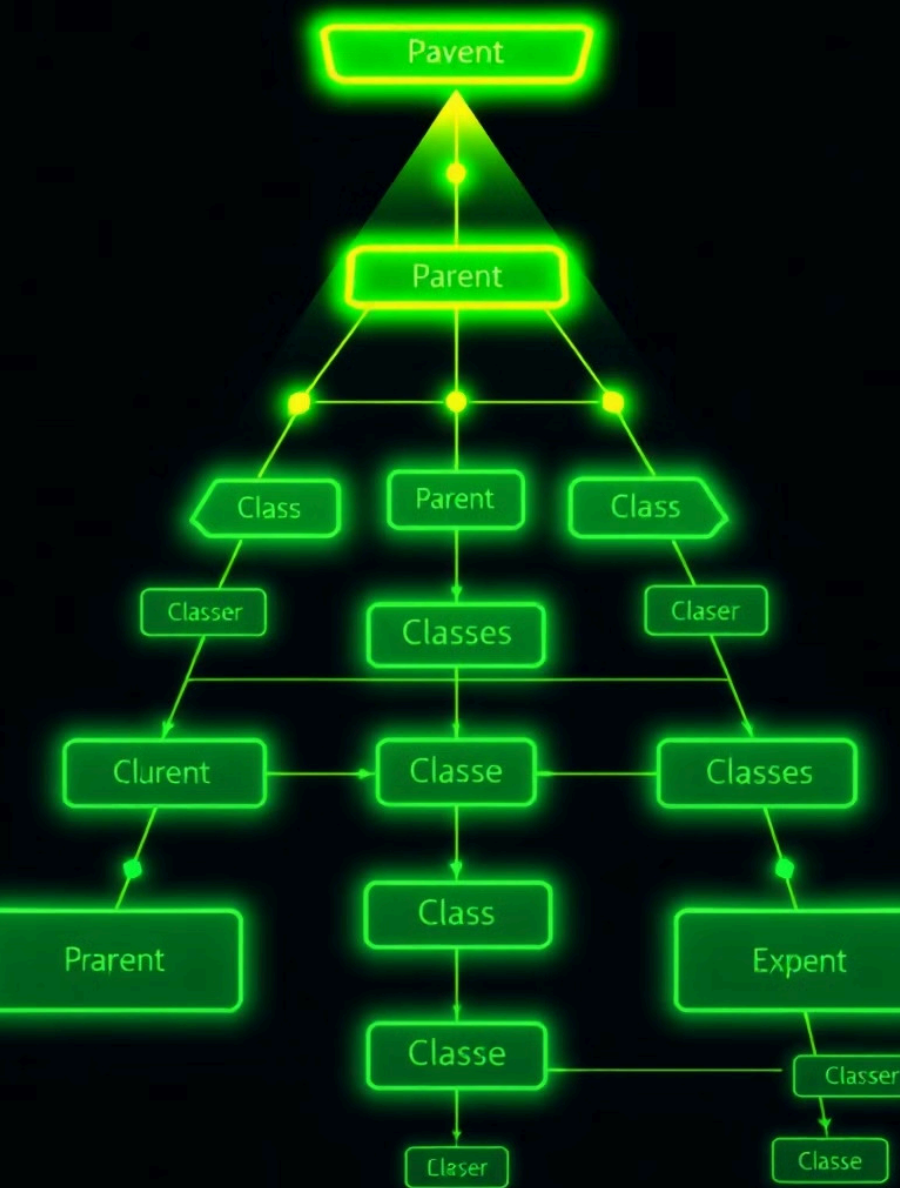
```
def obter_dados_usuario():
    tipo = input("Digite o tipo de forma (retangulo ou circulo): ").lower()
```

```
    if tipo == "retangulo":
        largura = float(input("Digite a largura do retângulo: "))
        altura = float(input("Digite a altura do retângulo: "))
        forma = Retangulo(largura, altura)
    elif tipo == "circulo":
        raio = float(input("Digite o raio do círculo: "))
        forma = Circulo(raio)
    else:
        print("Forma não reconhecida. Tente novamente.")
        return None

    return forma
```

```
if __name__ == "__main__":
    forma = obter_dados_usuario()
```

```
    if forma:
        print(f"\n{forma.__class__.__name__}:")
        print(f"Área: {forma.calcular_area()}")
        print(f"Perímetro: {forma.calcular_perimetro()}")
```



Herança: Reutilizando e Expandindo

Criando Novas Classes

Herança permite criar novas classes (classes filhas) baseadas em classes existentes (classes pais). Essa técnica facilita o reuso de código e a organização de classes em uma hierarquia.

Herdando Comportamento

As classes filhas herdam atributos e métodos da classe pai, mas podem também sobrescrevê-los para implementar comportamentos específicos. Isso permite a criação de classes especializadas com funcionalidades únicas, mas que compartilham características comuns.

Herança na Prática: Hierarquias de Classes



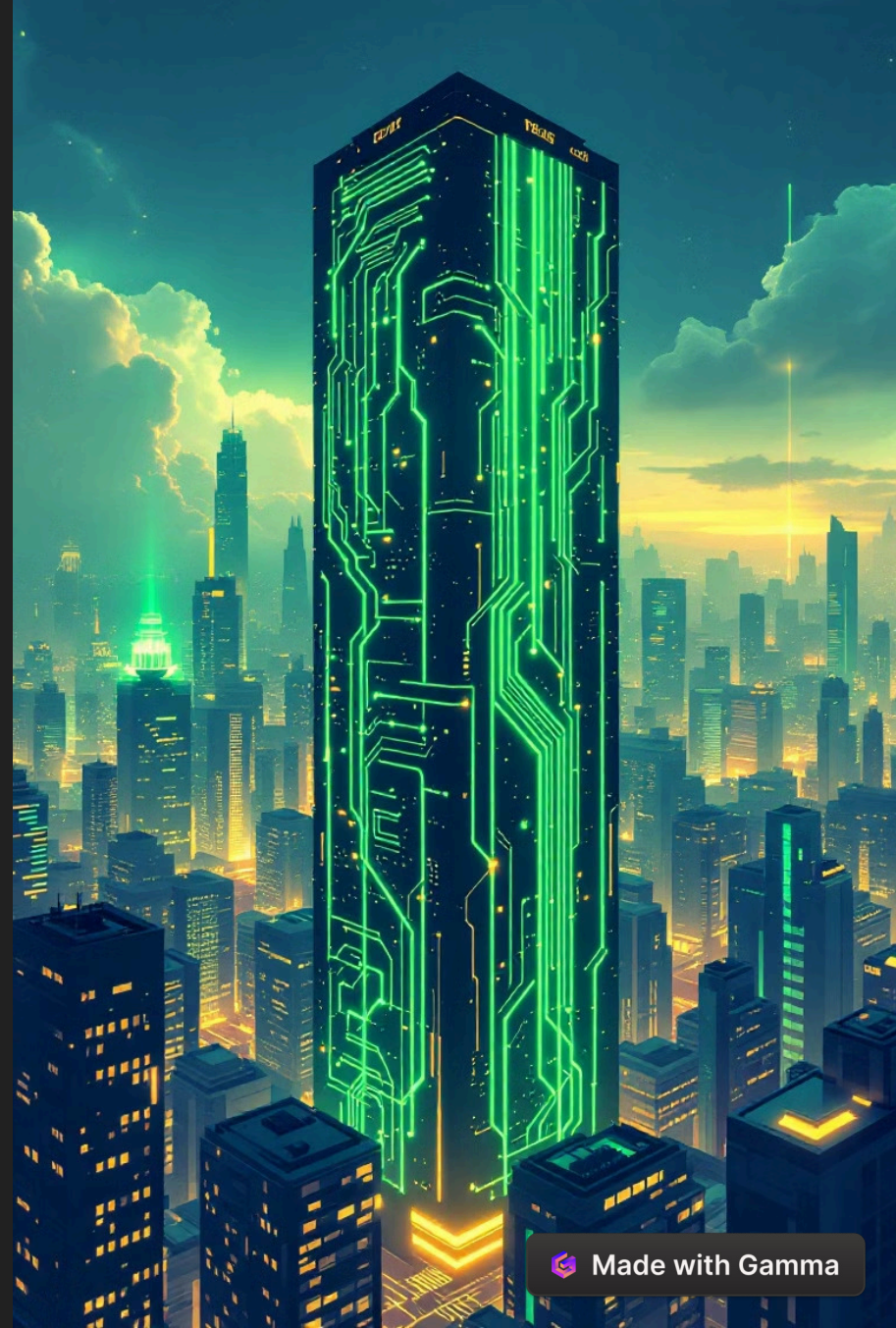
Hierarquias


Classes podem herdar de outras classes em diferentes níveis, formando uma hierarquia complexa. Essa estrutura permite a organização e o reuso de código em larga escala.



Tipos

Existem tipos de herança, como herança simples (uma classe filha herda de uma classe pai) e herança múltipla (uma classe filha herda de várias classes pais). A herança múltipla pode ser complexa e nem sempre é recomendada.





Polimorfismo: Muitas Formas, Uma Interface

1

O Poder da Flexibilidade

Polimorfismo significa a capacidade de um objeto assumir muitas formas. Permite que objetos de diferentes classes sejam tratados de forma uniforme através de uma interface comum. Isso simplifica o código e aumenta a flexibilidade.

2

Implementação e Interfaces

O polimorfismo é implementado através de herança e interfaces. Um método definido na classe pai pode ser sobrescrito por classes filhas para implementar o comportamento desejado.

Polimorfismo em Ação: Flexibilidade e Extensibilidade

1

Sobrecarga

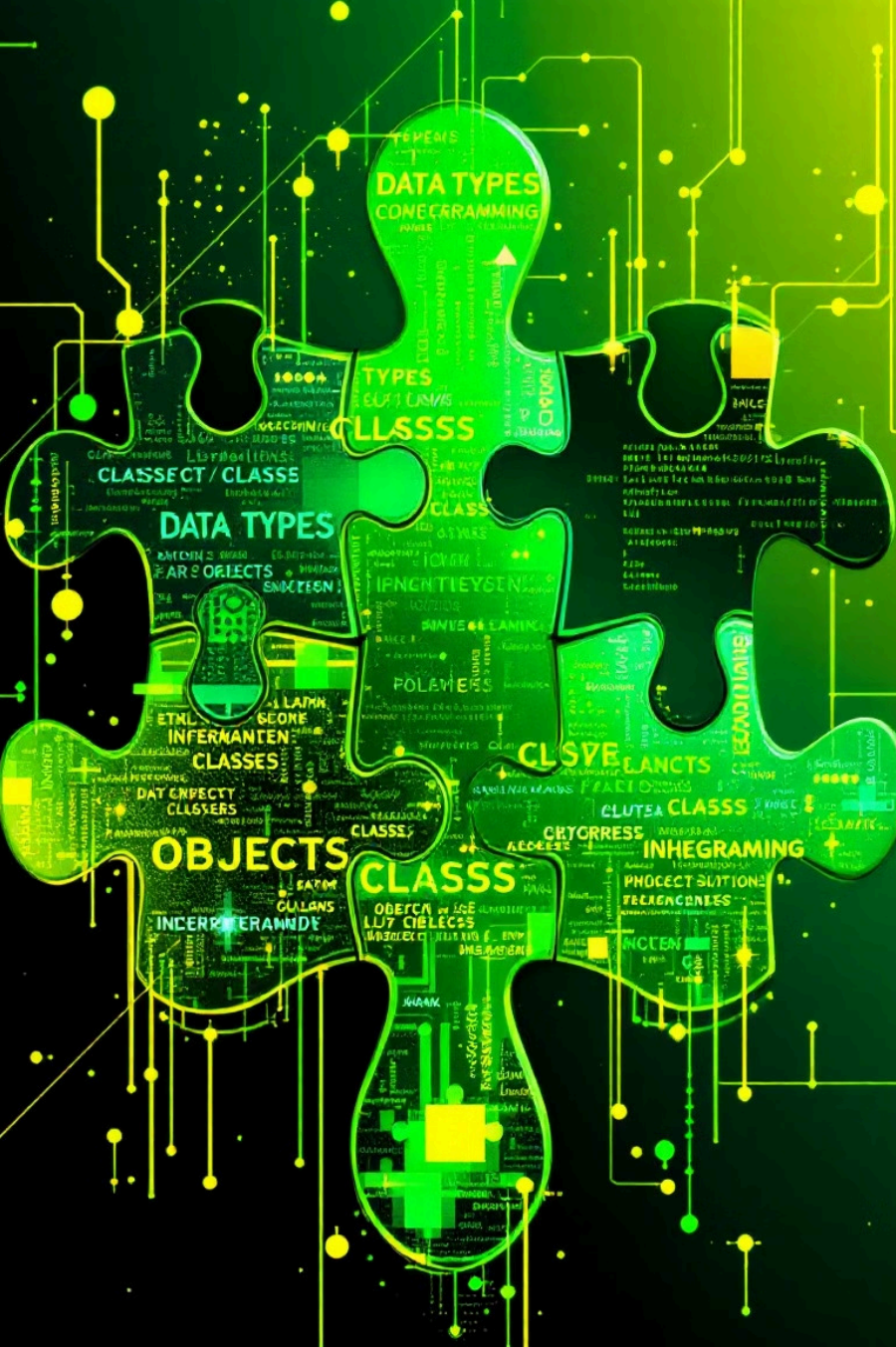
Sobrecarga de métodos permite que métodos com o mesmo nome, mas com diferentes parâmetros, sejam definidos em uma classe. Isso aumenta a flexibilidade e legibilidade do código.

2

Sobrescrita

Sobrescrita de métodos permite que classes filhas sobrescrevam métodos herdados da classe pai, implementando comportamentos específicos. Isso permite a criação de classes especializadas com funcionalidades únicas.





Conclusão: Dominando os Pilares da POO

A abstração, a herança e o polimorfismo são os pilares da POO. Eles permitem criar software mais organizado, modular e reutilizável. Domine esses conceitos para se tornar um desenvolvedor mais eficiente e prepare-se para construir aplicações incríveis!