Escola e Faculdade de Tecnologia Senai Roberto Mange.

Ana Claudia Lopes dos Santos

PROGRAMAÇÃO ORIENTADA A OBJETOS

Campinas

2025

Herança e Polimorfismo

1. **Conceito teórico de herança:**

   A herança permite que as classes compartilhes seus atributos, métodos e outros membros da classe entre si.

   Ela é usada com a intenção de reaproveitar código e de evitar a repetição excessiva dele.

   Na herança, existem dois tipos principais de classe que são:
   1. Classe Base: Concede características a uma outra
   2. Classe Derivada: Herda as características da classe base

2. **Conceito teórico de polimorfismo:**

   O polimorfismo pode ser definido como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar métodos que, embora apresentem a mesma assinatura, se comportam de maneira distinta.

   Basicamente, ele é um mecanismo que usamos para selecionar as funcionalidades utilizadas de forma dinâmica por um programa no decorrer da sua execução.

   Com ele, é possível atribuir o mesmo atributo a objetos diferentes.

3. **Exemplo prático de herança (Python):**

```
class Personagem:  # Definimos a classe Personagem
    def __init__(self, nome, vida):  # metodo construtor
__init__ e atribuímos os    parametros nome e vida
        self.nome = nome
        self.vida = vida


class Heroi(Personagem):  # Herói herda de Personagem
    def __init__(self, nome, vida, habilidade):
        super().__init__(nome, vida)  # Chamando o construtor
da classe mãe
        self.habilidade = habilidade
```

```
heroi1 = Heroi("Superman", 100, "Voo")  # Criando um herói
heroi2 = Heroi("Homem de Ferro", 200, "Morrer")
print(heroi1.nome)  # Saída: Superman
print(heroi1.vida)  # Saída: 100
print(heroi1.habilidade)  # Saída: Voo
```

4. **Exemplo prático de polimorfismo:**
   O exemplo a seguir, é a continuação do exemplo supracitado.

```
class Personagem:
  def __init__(self, nome, vida):
      self.nome = nome
      self.vida = vida


  class Heroi(Personagem):
      def __init__(self, nome, vida, habilidade):
          super().__init__(nome, vida)
          self.habilidade = habilidade


  class Vilao(Personagem):  # Adicionando a classe Vilão
      def __init__(self, nome, vida, poder):
          super().__init__(nome, vida)
          self.poder = poder


  def atacar(personagem):  # Função para atacar, pode ser
chamada por heróis ou vilões
      print(f"{personagem.nome} está atacando!")


  heroi1 = Heroi("Superman", 100, "Voo")
  vilao1 = Vilao("Lex Luthor", 80, "Inteligência")


  atacar(heroi1)  # Chamando a função atacar() com um herói
  atacar(vilao1)  # Chamando a função atacar() com um vilão
```

5. **Diferenças entre Herança e Polimorfismo:]**

A diferença principal entre os dois é o que cada um faz na programação orientada a objetos. A herança serve para criar classes usando outras que já existem,

tipo reaproveitando o código, sem precisar escrever tudo de novo toda hora. Ela meio que monta a base que o polimorfismo vai usar depois. O polimorfismo entra para fazer com que os métodos possam agir de jeitos diferentes, dependendo da situação. Resumindo: a herança decide o que a classe nova vai pegar da outra, e o polimorfismo muda como as coisas vão funcionar.

6. Vantagens da herança em POO

- **Aproveitar o código já feito**: Como uma classe pode pegar coisas de outra, dá para usar o que já foi criado sem ter que programar tudo de novo.
- **Melhorar o que já existe**: As classes que herdam de outras podem não só usar o que a classe antiga tem, mas também adicionar coisas novas e ficar ainda melhores.
- **Deixar tudo mais organizado**: Como as classes seguem uma ordem tipo em escada (hierarquia), o código fica mais fácil de entender e bem arrumado.

7. Vantagens do polimorfismo em POO

- **Reaproveitar o código**: Dá para usar o mesmo método em várias situações, e ele ainda pode agir de jeitos diferentes dependendo do caso.
- **Código mais claro**: A forma como tudo é organizado ajuda a entender melhor o que tá acontecendo no código.
- **Menos repetição**: Não precisa criar várias funções com nomes diferentes para cada coisa. Com o mesmo nome, dá para tratar vários objetos de um jeito só.

REFERÊNCIAS

https://kinsta.com/pt/blog/programacao-orientada-objetos-python/

Wellington. **Conceitos e Exemplos – Herança: Programação Orientada a Objetos – Parte 1.** 2010, Disponível em: **https://www.devmedia.com.br/conceitos-e-exemplos-heranca-programacao-orientada-a-objetos-parte-1/18579**

Acesso em: 20/05/2025

SILVA, Silas. **Herança e Polimorfismo em Python: Aprenda a Estruturar suas Hierarquias de Classes,** 2024, Disponível em: **https://www.dio.me/articles/heranca-e-polimorfismo-em-python-aprenda-a-estruturar-suas-hierarquias-de-classes**

Acesso em: 20/05/2025

Wellington, **Conceitos e Exemplos – Polimorfismo: Programação Orientada a Objetos.** 2010, Disponível em: https://www.devmedia.com.br/conceitos-e-exemplos-polimorfismo-programacao-orientada-a-objetos/18701

Acesso em: 20/05/2025