PESQUISA DETALHADA – DOIS PILARES MAIS RELEVANTES DA POO:

HERANÇA E POLIMORFISMO

Ketlyn Lorrayne Niza de Araújo – 1DS-TB

Professor Dorival

**Herança na Programação Orientada a Objetos:**

É responsável por permitir que uma classe compartilhe atributos e métodos através de “heranças”. É utilizada para reaproveitar códigos, comportamentos generalizados, ou especializar operações ou atributos. Ela cria uma relação entre classes em forma de hierarquia, como se fosse uma arvore, podendo ser mais genérica e outras mais especificas.

Neste processo existem dois tipos de classe:

* **Classe base (ou superclasse)**: é a classe "mãe", que define os comportamentos e atributos comuns.
* **Classe derivada (ou subclasse)**: é a classe "filha", que herda tudo da classe mãe, mas pode adicionar coisas novas ou modificar o que for necessário.

Quando utilizamos herança, o código tende a ficar mais organizado e de fácil compreensão, como também facilita seu crescimento, pois não é necessário repetir diversas vezes a mesma coisa, sendo somente necessário valores diferentes para as classes filhas.

**Vantagens:**

- Reutilização de código: a classe filha herda da classe pai o conteúdo, evitando a necessidade de reescrita do mesmo código várias vezes.

- Manutenção facilitada: Alterações feitas na classe pai são automaticamente refletidas nas classes filhas, tornando a manutenção do código mais simples e eficiente.

- Organização do código: a herança cria uma estrutura hierárquica, facilitando a organização e compreensão do relacionamento entre diferentes classes.

Exemplo:

class Pessoa:  
 def \_\_init\_\_(self, nome):  
 self.nome = nome  
  
class Aluno(Pessoa):  
 pass  
  
aluno = Aluno("Kety")  
print(aluno.nome) # Saída: Kety

**Polimorfismo na Programação Orientada a Objetos:**

É a capacidade de objetos de diferentes classes responderem a mesma mensagem de maneiras distintas, ou seja, objetos de classes diferentes podem ter métodos com o mesmo nome, mas com comportamentos diferentes, o que permite ao código trabalhar com objetos de tipos diferentes de forma uniforme, utilizando a mesma interface ou método.

**Exemplo:**

Imagina que você tem uma classe chamada Animal, que diz que todo animal precisa "falar". Mas ela não define como, apenas avisa que esse método vai existir.

Depois, você cria duas classes: Cao e Gato. Elas herdam de Animal e cada uma implementa o método falar () do seu jeito:

* O Cao fala "Au au!"
* O Gato fala "Miau!"

Exemplo python:

class Animal:  
 def falar(self):  
 pass  
  
class Cao(Animal):  
 def falar(self):  
 return "Au au!"  
  
class Gato(Animal):  
 def falar(self):  
 return "Miau!"

Vantagens:

* **Reutilização de código:** um mesmo método pode ser usado em várias classes diferentes, com comportamentos específicos, evitando repetição desnecessária.
* **Facilidade na manutenção:** se for preciso ajustar um comportamento, você faz isso direto na classe certa, sem mexer no resto do sistema.
* **Flexibilidade:** novas classes podem ser criadas com o mesmo método, adaptando a lógica sem precisar alterar o que já existe.
* **Organização:** o código fica mais limpo e estruturado, facilitando a leitura e o entendimento.
* **Tratamento personalizado:** cada classe pode responder de forma diferente a um mesmo método, permitindo respostas específicas para cada caso.

Herança X Poliformismo

| **Aspecto** | **Herança** | **Polimorfismo** |
| --- | --- | --- |
| **O que é** | Uma classe filha herda atributos e métodos de uma classe pai | Classes diferentes usam o mesmo método com comportamentos diferentes |
| **Reutilização de código** | Sim, evita reescrever o mesmo código em várias classes | Sim, reutiliza a lógica do método com variações específicas |
| **Organização** | Cria uma estrutura hierárquica (ex: Pai → Filho) | Dá mais liberdade e flexibilidade no uso dos métodos |
| **Flexibilidade** | Boa para reaproveitar o que já existe | Alta: o mesmo método se adapta a diferentes tipos |
| **Manutenção** | Fácil: mudar na classe pai afeta todas as filhas | Fácil: muda o comportamento só onde for necessário |
| **Exemplo** | Aluno herda de Pessoa | falar() é diferente em Cachorro e Gato, mesmo nome, ações diferentes |
| **Uso ideal** | Quando várias classes compartilham características e comportamentos iguais | Quando diferentes classes precisam ter comportamentos diferentes com o mesmo nome |

**Referências:**

* <https://www.inf.pucrs.br/~michael/aulas/algo_progII/apresentacoes/Heranca-Polimorfismo-MetAbstrato.htm>
* <https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/object-oriented/inheritance>
* <https://docs.python.org/3/tutorial/classes.html>