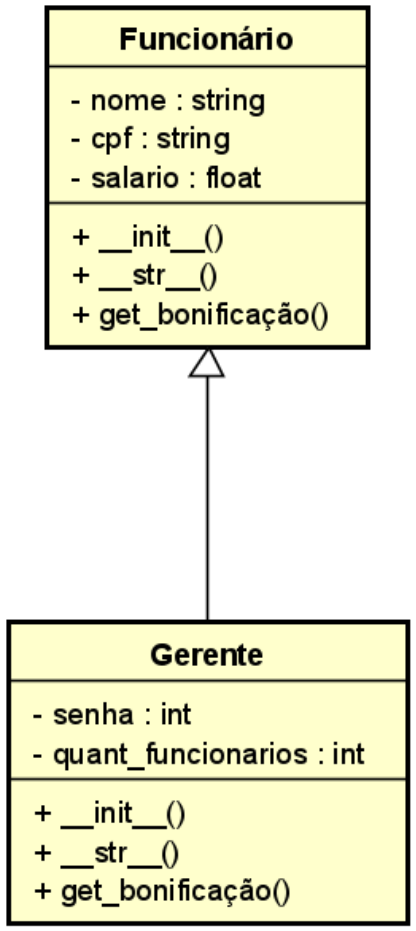
Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo

- O conceito de **polimorfismo** significa que podemos tratar instâncias de classes diferentes da mesma maneira.
- É a capacidade de um objeto poder ser referenciado de várias formas
- Em outras palavras, podemos enviar uma mensagem a um objeto sem saber seu tipo a priori e deixar que o compilador resolva em tempo de execução.

- Polimorfismo não quer dizer que o objeto fica se transformando, muito pelo contrário, um objeto nasce de um tipo e morre daquele tipo, o que pode mudar é a maneira como nos referimos a ele

Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo



O que guarda uma variável do tipo Funcionário?

R- Uma referência para um Funcionário , nunca o objeto em si.

Pela herança, todo Gerente é um funcionário, pois é uma extensão deste.

Podemos nos referir a um Gerente como sendo um Funcionário.

Se alguém quiser falar com um Funcionário do Banco, pode falar com o Gerente?

Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo

Percebemos que além dos métodos habituais: __init__ e __str__, o método **get_bonificação**() que inicialmente foi definido na classe-pai Funcionário, foi reescrito na classe-filha Gerente.

Justificativa: Todo fim de ano, os funcionários do nosso banco recebem uma bonificação. Os funcionários comuns recebem 10% do valor do salário e os gerentes, 15%.

```python
class Funcionario:
    def __init__(self, nome, cpf, salario):
        self.__nome = nome
        self._cpf = cpf
        self._salario = salario

    def get_bonificacao(self):
        return self._salario * 0.10
    @property
    def nome(self):...

class Gerente(Funcionario):
        def __init__(self, nome, cpf, salario, senha, tamanho):
            super().__init__(nome, cpf, salario)
            self.__senha = senha
            self.tamanho_equipe = tamanho

        def get_bonificacao(self):
            return self._salario * 0.15
```

Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo

```
joão = Funcionario('João', '111111111-11', 2000.0)
josé = Gerente('José', '222222222-22', 5000.0, '1234', 20)

print("A bonificação do {} foi de:{}".format(joão.nome,joão.get_bonificacao()))
print("A bonificação do {} foi de:{}".format(josé.nome,josé.get_bonificacao()))
```

Até ai tudo bem! Mas cadê o polimorfismo nisso?

Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo

- Na verdade o polimorfismo é implementado através de um método polimórfico.
- E para implementarmos um método polimórfico precisamos ter uma relação de herança entre as classes e pelo menos um método reescrito.

Suponhamos que as bonificações de todos os funcionários seja controlada por uma classe: ControleDeBonificações

```python
from classes3 import *
class ControleDeBonificacoes:
    def __init__(self, total_bonificacoes=0):
        self._total_bonificacoes = total_bonificacoes

    def registra(self, funcionario):
        self._total_bonificacoes += funcionario.get_bonificacao()
    @property
    def total_bonificacoes(self):
        return self._total_bonificacoes

joão = Funcionario('João', '111111111-11', 2000.0)
josé = Gerente("José", "222222222-22", 5000.0, '1234', 20)
controle = ControleDeBonificacoes()
controle.registra(joão)
controle.registra(josé)
print("total pago em bonificações: {}".format(controle.total_bonificacoes))
```

Método polimórfico

Programação Orientada à Objetos: Reutilização de Classes: Polimorfismo

Exercício

Melhorando a classe ControledeDeBonificações acrescentando um atributo que representa uma lista de funcionários...

https://colab.research.google.com/drive/100pgaUnQR0o6fi4AhjNvrBSHpLL4KHMC?usp=sharing