



IN

dev full stack- Python
Aula 7 - Polimorfismo

Revisão

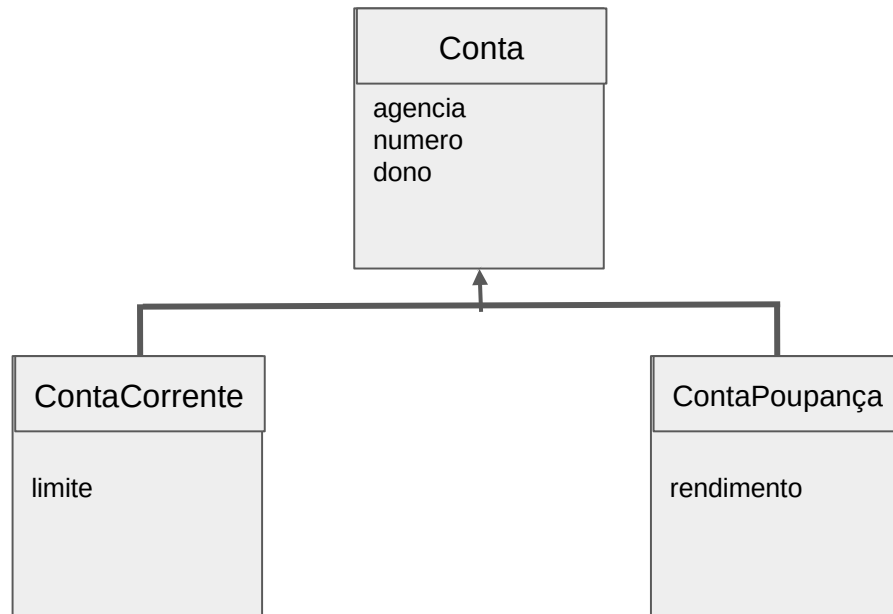
Herança

—



Herança

- Conceito de Orientação a Objetos
- Uma classe pode herdar atributos e métodos de uma outra classe
- Classe filha x Classe mãe



Situação

- Funcionário e Gerente

Gerente
nome cpf salario senha qtd_gerenciados

Funcionário
nome cpf salario

**Atributos comuns:**

- nome
- cpf
- salario

Situação

- Funcionário e Gerente

Gerente
nome cpf salario senha qtd_gereciados

Funcionário
nome cpf salario

Funcionario.py > ...

```
1 class Funcionario:
2     def __init__(self, nome, cpf, salario):
3         self.__nome = nome
4         self.__cpf = cpf
5         self.__salario = salario
6
```

Gerente.py > ...

```
1 class Gerente:
2     def __init__(self, nome, cpf, salario, senha, qt_gereciados):
3         self.__nome = nome
4         self.__cpf = cpf
5         self.__salario = salario
6         self.__senha = senha
7         self.__qt_gereciados = qt_gereciados
8
```

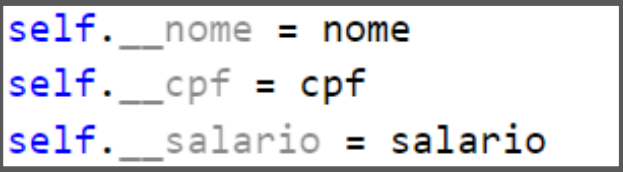
Situação

- Funcionário e Gerente

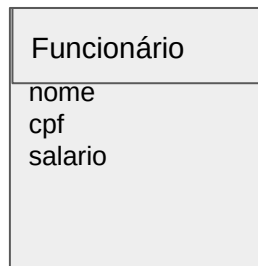
Repetindo...

Gerente.py > ...

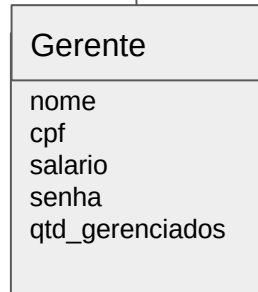
```
1  class Gerente:
2      def __init__(self, nome, cpf, salario, senha, qt_gerenciados):
3          self.__nome = nome
4          self.__cpf = cpf
5          self.__salario = salario
6          self.__senha = senha
7          self.__qt_gerenciados = qt_gerenciados
8
```



Situação - Resolvendo



Superclasse



Subclasse

Situação - Resolvendo (no código)

```
class Gerente(Funcionario):  
    def __init__(self, nome, cpf, salario, senha, qt_gerenciados):  
        super().__init__(nome, cpf, salario)  
        self.__senha = senha  
        self.__qt_gerenciados = qt_gerenciados
```

Herança

- Classe Gerente herda atributos e métodos da classe Funcionário

```
1  class Funcionario:
2      def __init__(self, nome, cpf, salario):
3          self._nome = nome
4          self._cpf = cpf
5          self._salario = salario
6
7      def salarioLiquido(self, itens_descontos):
8          liquido = self._salario
9          for desconto in itens_descontos:
10             liquido -= desconto
11         return liquido
```



Herança

- Usa-se apenas um `_` para protected
- Protected: Classes filhas (subclasses) podem acessar os atributos da classe mãe(superclasse) como se fossem seus.

```
1  class Funcionario:
2      def __init__(self, nome, cpf, salario):
3          self._nome = nome
4          self._cpf = cpf
5          self._salario = salario
6
```

Herança

main.py > ...

- ```
1 from Gerente import Gerente
2
3 gerente = Gerente("Ana", "4455577", 25000.00, "654321", 12)
4 plano_saude = 300
5 inss = 280
6 i_r = 500
7 lista_descontos = {plano_saude, inss, i_r}
8 salario_liquido = gerente.salarioLiquido(lista_descontos)
9 print(salario_liquido)
```

# Herança - Reescrita de Método

- Escrever especificamente qual o comportamento em uma classe específica

## Funcionario.py

```
def bonifica(self):
 return self._salario * 0.1
```

## Gerente.py

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

# Herança - Reescrita de Método

main.py > ...

```
1 from Gerente import Gerente
2 from Funcionario import Funcionario
3
4 gerente = Gerente("Ana", "4455577", 25000.00, "654321", 12)
5 plano_saude = 300
6 inss = 280
7 i_r = 500
8 lista_descontos = {plano_saude, inss, i_r}
9 salario_liquido = gerente.salarioLiquido(lista_descontos)
10 print(salario_liquido)
11
12 print(gerente.bonifica())
```

**3.750,00**

# Exercício

Desenvolver um sistema capaz de cadastrar médicos com CRM, Nome, Idade e Salário. O médico pode ser aposentado, para isto, deveremos verificar no geral, se ele tem mais de 55 anos. Deveremos verificar também o valor da aposentadoria dele, que será normalmente 80% do valor do salário dele. Quando o médico é auxiliar ele só se aposentará com 60 anos. Diferente de se o médico for cirurgião, que será com 50. Quando o médico é cirurgião também, o valor da aposentadoria dele tem, além dos 80% do salário um acréscimo de 2000 reais.

# Exercício



Desenvolver um sistema que auxilie no cadastro de instrumentos musicais de uma escola de música. Os instrumentos devem ser cadastrados com nome, grau de dificuldade e o professor que ensina o instrumento. O professor será cadastrado no sistema com nome e pontuação (de 1 a 10). Os instrumentos serão divididos por tipo, pois eles podem ser instrumento de corda, sopro ou percussão. Quando o instrumento é de corda, pode-se inserir informações da quantidade de cordas e tipo de corda. Já quando o instrumento é de percussão, pode ser informado se usa baqueta ou não. O grau de dificuldade do instrumento é calculado em primeiro lugar pela pontuação do professor que a leciona. Mas se o instrumento for de corda, se a corda for de aço, é multiplicado por quantidade de cordas pelo grau de dificuldade.



# Polimorfismo

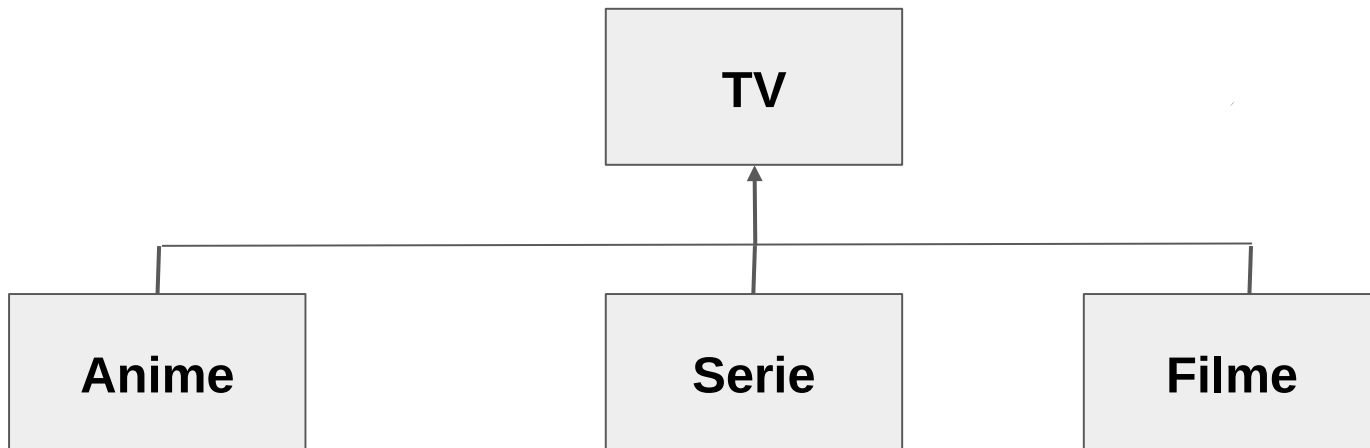


# Polimorfismo

- É a propriedade de duas ou mais classes derivadas de uma mesma superclasse responderem a mesma mensagem, cada uma de uma forma diferente.
- Ocorre quando uma subclasse redefine um método existente na superclasse, ou seja, quando temos os **métodos sobrescritos** (overriding).

# Polimorfismo

- É a propriedade de duas ou mais classes derivadas de uma mesma superclasse responderem a mesma mensagem, cada uma de uma forma diferente.
- Ocorre quando uma subclasse redefine um método existente na superclasse, ou seja, quando temos os **métodos sobrescritos** (overriding).



# Polimorfismo

## Classe mãe

```
Tv.py > ...
1 class Tv:
2 def __init__(self, nome, personagens, duracao):
3 self._nome = nome
4 self._personagens = personagens
5 self._duracao = duracao
6
7 @property
8 def nome(self):
9 return self._nome
10 @nome.setter
11 def nome(self, nome):
12 self._nome = nome
13 @property
14 def personagens(self):
15 return self._personagens
16 @personagens.setter
17 def personagens(self, personagens):
18 self._personagens = personagens
19 @property
20 def duracao(self):
21 return self._duracao
22 @duracao.setter
23 def duracao(self, duracao):
24 self._duracao = duracao
```

# Polimorfismo

Anime.py > ...

```
1 from Tv import Tv
2 class Anime(Tv):
3 def __init__(self, nome, personagens, duracao, studio):
4 super().__init__(nome, personagens, duracao)
5 self.__studio = studio
6
7 @property
8 def studio(self):
9 return self.__studio
10 @studio.setter
11 def studio(self, studio):
12 self.__studio = studio
```

## Classe Filha 1

# Polimorfismo

📁 Serie.py > ...

```
1 from Tv import Tv
2
3 class Serie(Tv):
4 def __init__(self, nome, personagens, duracao, quantidade_episodios):
5 super().__init__(nome, personagens, duracao)
6 self.__quantidade_episodios = quantidade_episodios
7
8 @property
9 def quantidade_episodios(self):
10 return self.__quantidade_episodios
11 @quantidade_episodios.setter
12 def quantidade_episodios(self, quantidade_episodio):
13 self.__quantidade_episodios = quantidade_episodio
```

## Classe Filha 2

# Polimorfismo

Filme.py > ...

```
1 from Tv import Tv
2
3 class Filme(Tv):
4 | def __init__(self, nome, personagens, duracao):
5 | | super().__init__(nome, personagens, duracao)
6
```

## Classe Filha 3

# Polimorfismo

Tv.py > ...

```
26 def aluga(self):
27 pass
```

Anime.py > ...

```
13
14 def aluga(self):
15 return 2.5
16
```

Serie.py > ...

```
14
15 def aluga(self):
16 return self.__quantidade_episodios * 1.9
17
```

Filme.py > ...

```
6
7 def aluga(self):
8 return 1.5
9
```



# Polimorfismo

```
main.py > ...
1 from Tv import Tv 2.5
2 from Anime import Anime 1.5
3 from Filme import Filme 13.299999999999999
4 from Serie import Serie
5
6 personagens1 = ["Meliodas", "Diane", "Elisabeth"]
7 anime = Anime("Nanatsu No Taizai", personagens1, 480, "Deen")
8
9 personagens2 = ["Morpheus", "Agente Smith", "Neo"]
10 filme = Filme("Matrix", personagens2, 120)
11
12 personagens3 = ["Annie", "Gilbert", "Diana"]
13 serie = Serie("Annie with an E", personagens3, 500, 7)
14
15 print(anime.aluga())
16 print(filme.aluga())
17 print(serie.aluga())
18
```

# Classe Abstrata

- Classe que não permitem realizar qualquer tipo de instância
- São classes feitas especialmente para serem modelos para suas classes derivadas
- Obs: Não temos “nativo” no python

# Classe Abstrata


```
Traceback (most recent call last):
 File "c:\Users\frana\Documents\Python\TV\main.py", line 10, in <module>
 filme = Filme("Matrix", personagens2, 120)
TypeError: Can't instantiate abstract class Filme with abstract method aluga
```

- Biblioteca: abc

Tv.py > ...

```
6 @abc.abstractmethod
7 def aluga(self):
8 pass
9
```

Obriga as classes filhas  
implementarem este  
método



# Classe Abstrata

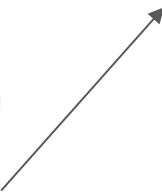
```
Traceback (most recent call last):
 File "c:\Users\frana\Documents\Python\TV\main.py", line 10, in <module>
 filme = Filme("Matrix", personagens2, 120)
TypeError: Can't instantiate abstract class Filme with abstract method aluga
```

- Biblioteca: abc

Tv.py > ...

```
6 @abc.abstractmethod
7 def aluga(self):
8 pass
9
```

Obriga as classes filhas  
implementarem este  
método



```
tv = Tv(100, 100, personagens, 20)
TypeError: Can't instantiate abstract class Tv with abstract method aluga
```

# Classe Abstrata

- Classes abstratas não podem ser instanciadas.

```
personagens1 = ["Meliodas", "Diane", "Elisabeth"]
anime = Anime("Nanatsu No Taizai", personagens1, 480, "Deen")
```

```
personagens2 = ["Morpheus", "Agente Smith", "Neo"]
filme = Filme("Matrix", personagens2, 120)
```

```
personagens3 = ["Annie", "Gilbert", "Diana"]
serie = Serie("Annie with an E", personagens3, 500, 7)
```

```
tv = Tv("Teste", personagens1, 50)
```

```
print(anime.aluga())
print(filme.aluga())
```

TypeError: Can't instantiate abstract class Tv with abstract method aluga

# Atividade

Em um campeonato de surf podem ter 3 categorias:

- Amador
- Profissional
- Lenda

Para todos os tipos de campeonato, temos o cadastro do nome do campeonato, local onde ocorrerá, premiação, patrocinadores (previamente cadastrados com nome e valor) e os atletas. Os atletas terão nome, idade, pontuação e categoria que compete (Amador, profissional ou lenda). Os atletas lenda podem participar de qualquer competição, já os profissionais, apenas do profissional e do amador, sendo que o amador só poderá participar do próprio circuito. É preciso haver este bloqueio no sistema. A pontuação do atleta que ganha o campeonato é:

- Amador: 10 pontos
- Profissional: 50 pontos
- Lenda: 100 pontos

# Atividade

- Os alunos de uma universidade têm acesso a diversos materiais de estudo na biblioteca virtual: apostilas, livros e podcasts. Todos eles cadastrados com nome, assuntos e tempo de estudo. É estimado um tempo de estudo de 30min para cada assunto de livro, 15min de apostila e 5min de podcast. O podcast vai ter opção de ser colocado em uma velocidade 2x, se este for o caso, o tempo será calculado na metade: 2.5 min. Crie um sistema em python para criar estas regras e testá-las.

The logo consists of the letters 'IN' in a white, serif font, centered within a solid red square. The background of the entire image is a dark red with abstract, layered geometric shapes in lighter shades of red, creating a sense of depth and movement.

# IN

71 3901 1052 | 71 9 9204  
0134

@infinity.school

[www.infinityschool.com.br](http://www.infinityschool.com.br)

Salvador Shopping Business | Torre Europa Sala  
310 Caminho das Árvores, Salvador - BA CEP: