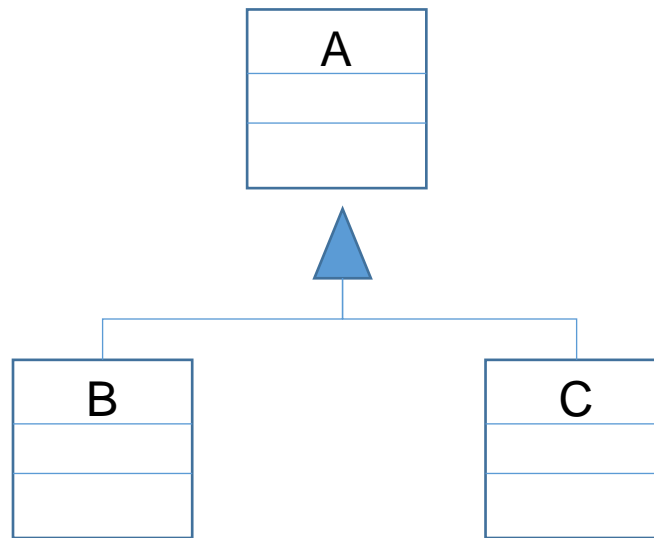


Programação Orientada à Objetos:

Reutilização de Classes: Herança

- Nem sempre precisamos escrever classes do zero.
- Se a classe que estivermos escrevendo for uma versão especializada de uma classe já existente, podemos utilizar uma outra forma de reuso chamada: **Herança**.

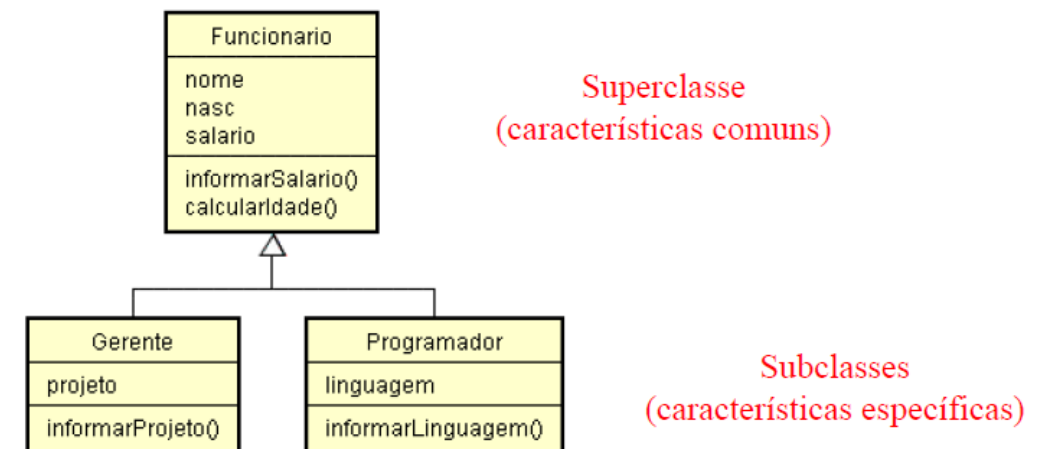
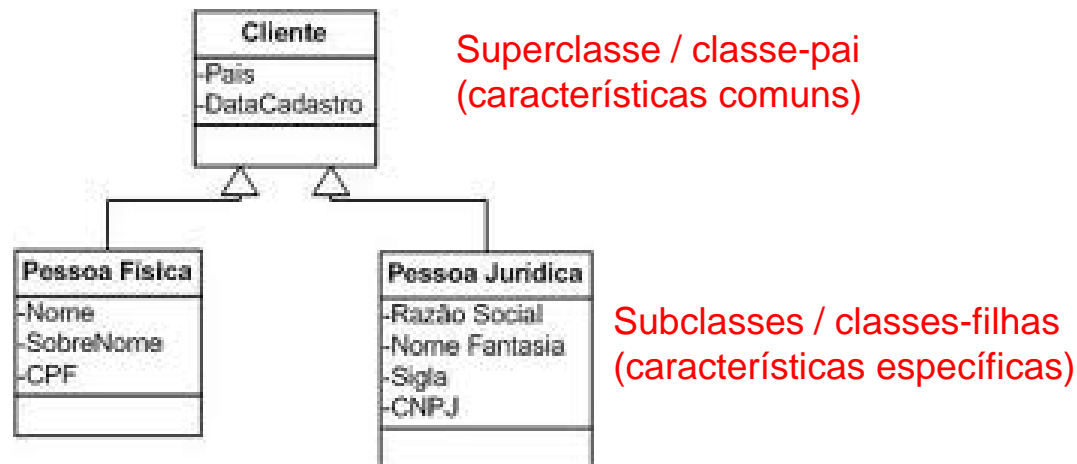
REPRESENTAÇÃO UML:



Programação Orientada à Objetos:

Reutilização de Classes: Herança

- Quando uma classe herda de outra, ela assumirá automaticamente todos os atributos e métodos da classe original, também conhecida como classe ancestral ou classe-pai.
- A nova classe herdada é chamada de classe-filha.
- A classe-filha, embora herde todos os atributos e métodos da classe-pai, é livre para criar novos atributos e métodos.



Programação Orientada à Objetos: Reutilização de Classes: Herança

- Representação em python:

```
class nome_da_classe (classe_pai_1, classe_pai_2,...):  
    atributos  
    métodos
```

```
class Cliente:
    def __init__(self, pais, DataCadastro):
        self.pais = pais
        self.DataCadastro = DataCadastro
```

Definição de Herança: PessoaFisica herda da classe Cliente

```
class PessoaFisica(Cliente):
    def __init__(self, nome, sobrenome, cpf):
        self.nome = nome
        self.sobrenome = sobrenome
        self.cpf = cpf
```

Definição de Herança: PessoaJuridica herda da classe Cliente

```
class PessoaJuridica(Cliente):
    def __init__(self, razaosocial, nomefantasia, cnpj):
        self.razaosocial = razaosocial
        self.nomefantasia = nomefantasia
        self.cnpj = cnpj
```

Programação Orientada à Objetos: Reutilização de Classes: Herança

- Tanto PessoaFísica quanto PessoaJurídica são extensões/especializações da classe Cliente
- Cada uma herda os atributos da superclasse e define seus atributos específicos
- A herança tem que ser explícita da subclasse para a superclasse, ou seja, é na definição das subclasses PessoaFísica/PessoaJuridica que teremos que dizer quem é a superclasse herdada.

Programação Orientada à Objetos:

Reutilização de Classes: Herança

- Precisamos garantir então que todo objeto criado do tipo PessoaFísica/PessoaJurídica vai herdar os atributos definidos na classe Cliente, pois toda PessoaFísica/PessoaJurídica é um Cliente.

```
maria = PessoaFisica("Maria", "Pereira", "2837192839")  
print(maria.pais)
```

Run: classes_herança x

```
C:\Users\roger\PycharmProjects\P00\venv\Scripts\python.exe C:/Users/roger/PycharmPro  
Traceback (most recent call last):  
  File "C:/Users/roger/PycharmProjects/P00/classes_herança.py", line 20, in <module>  
    print(maria.pais)  
AttributeError: 'PessoaFisica' object has no attribute 'pais'  
  
Process finished with exit code 1
```

Programação Orientada à Objetos:

Reutilização de Classes: Herança

- A forma de garantir isto é através do construtor da classe, ou seja, passamos também os atributos que todo Cliente tem para PessoaFisica e PessoaJuridica.
- Depois chamamos o construtor da superclasse para inicializar os atributos herdados através da palavra-chave: **super()**
- **super()** é utilizado toda vez que quisermos nos referir a um método de uma classe-pai ou superclasse.

Programação Orientada à Objetos:

Reutilização de Classes: Herança

```
class PessoaFisica(Cliente):
```

```
    def __init__(self, pais, DataCadastro, nome, sobrenome, cpf):
```

```
        super().__init__(pais, DataCadastro) ←
```

```
        self.nome=nome
```

```
        self.sobrenome=sobrenome
```

```
        self.cpf=cpf
```

```
class PessoaJuridica(Cliente):
```

```
    def __init__(self, pais, DataCadastro, razaosocial, nomefantasia, cnpj):
```

```
        super().__init__(pais, DataCadastro) ←
```

```
        self.razaosocial=razaosocial
```

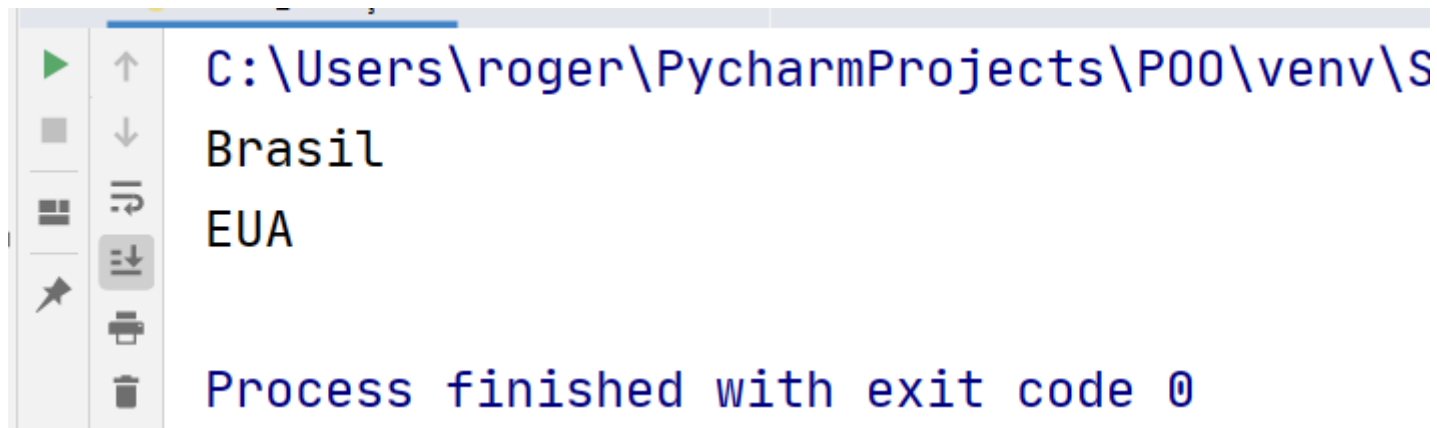
```
        self.nomefantasia=nomefantasia
```

```
        self.cnpj=cnpj
```


Programação Orientada à Objetos:

Reutilização de Classes: Herança

```
maria = PessoaFisica("Brasil", "12/12/2020", "Maria", "Pereira", "2837192839")
google = PessoaJuridica("EUA", "12/01/2005", "Google", "Google", "98342874283429")
print(maria.pais)
print(google.pais)
```



```
C:\Users\roger\PycharmProjects\P00\venv\S
Brasil
EUA
Process finished with exit code 0
```

Programação Orientada à Objetos: Reutilização de Classes: Herança

- **REESCRITA DE MÉTODOS**
- É comum em herança termos métodos de mesmo nome em mais de uma classe na hierarquia de classes. Como proceder?
- No Python, quando herdamos um método, podemos alterar seu comportamento. Podemos reescrever (sobrescrever, override) este método, assim como fizemos com o `__init__`.

Vamos ver um exemplo prático...