

Modelagem de Sistema de Gerenciamento de Projetos

Docente: André Ribeiro

Componentes da Equipe:



Jacqueline Navarro

</>jacquelinanavarro

Nikolas Martins

</>Salokled

Thaís de Souza

</>thaisdesouzabm

Vinicio Soares

</>vinicio5oares

Descrição do Cenário



- Uma empresa de software deseja implementar um sistema para gerenciar seus **projetos, funcionários e alocações de projetos**.
- Cada **projeto** tem um código, nome e data de início.
- Os **funcionários** têm um número de identificação, nome e cargo.
- Cada **alocação de projeto** deve registrar o funcionário alocado, o projeto em que ele está trabalhando e a data de início da alocação.

Tarefas solicitadas:

- Criar **classes** e **construtores** das entidades.
- Criar **instâncias** e realizar **testes**.

Diagrama Entidade Relacionamento - DER

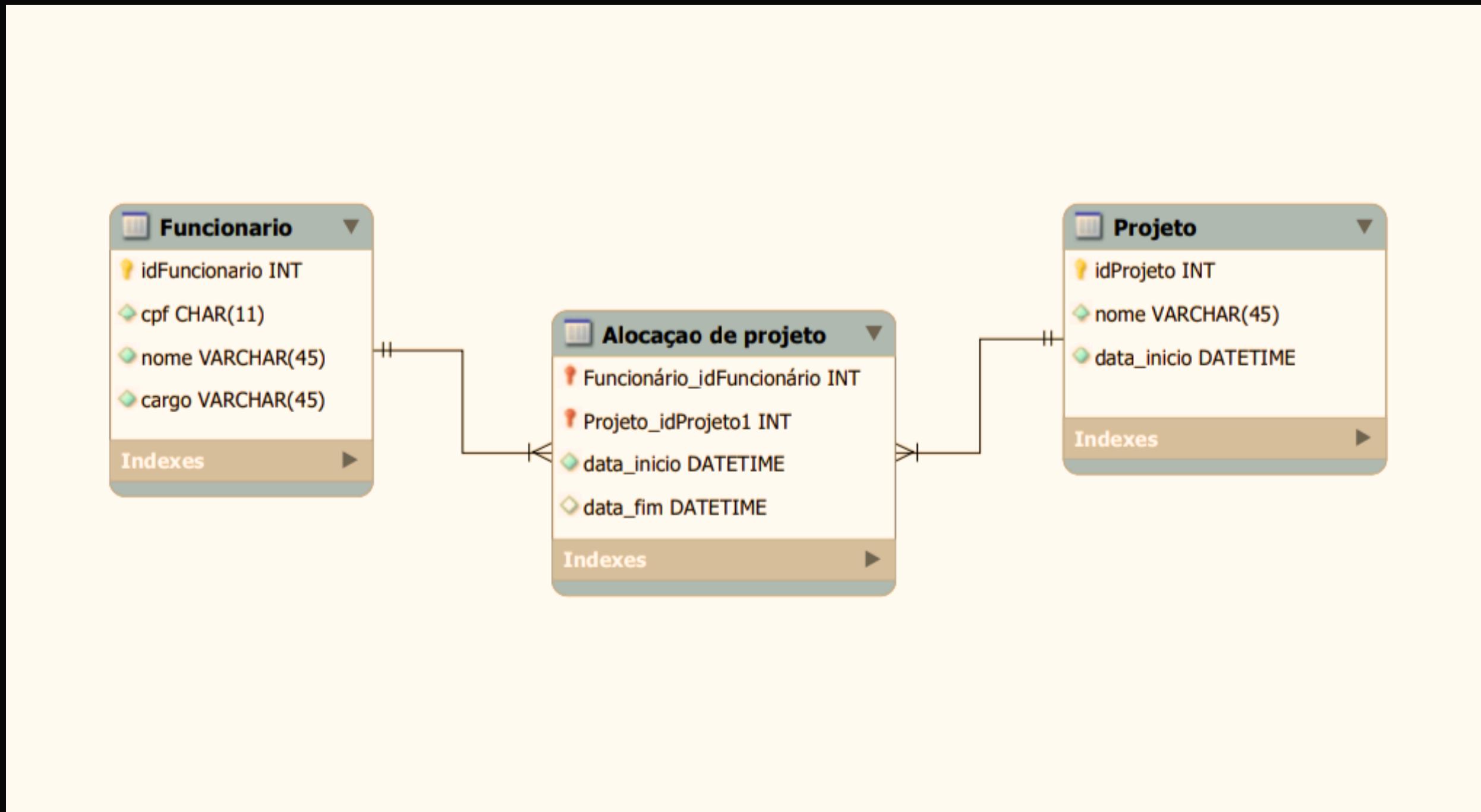
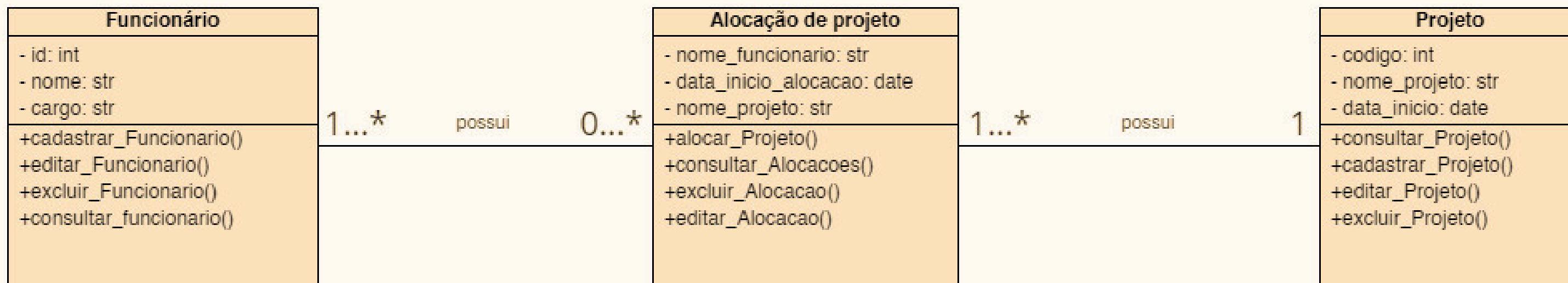


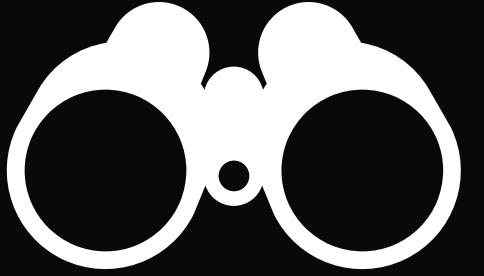
Diagrama de Classes



- O **funcionário** pode ter 0 projetos alocados ou N.
- A **alocação do projeto** pode ter 1 funcionário ou N.
- A **alocação do projeto** está ligada a apenas 1 projeto.
- O **projeto** pode ter 1 ou várias alocações (vários funcionários fazendo funções distintas).



Encapsulamento & Visibilidade no Código



O **encapsulamento** controla a **visibilidade** dos *atributos* e *métodos*, determinando quem pode acessá-los.

Os modificadores de acesso são indicados por convenções:

- *Atributos/métodos públicos* não têm prefixo especial.
- *Atributos/métodos protegidos* têm um único sublinhado no início
Exemplo: _atributo).
- *Atributos/métodos privados* têm dois sublinhados no início.
Exemplo: _atributo).

Encapsulamento



Atributos Privados: Na definição das classes ***Funcionário***, ***Projeto*** e ***Alocação***, os atributos são definidos como privados usando dois sublinhados (_) antes do nome do atributo. Isso significa que esses atributos não podem ser acessados diretamente fora da classe.

Exemplo:

```
class Funcionario:  
    def __init__(self, id, nome, cargo, genero):  
        self.__id = id  
        self.__nome = nome  
        self.__cargo = cargo  
        self.__genero = genero
```

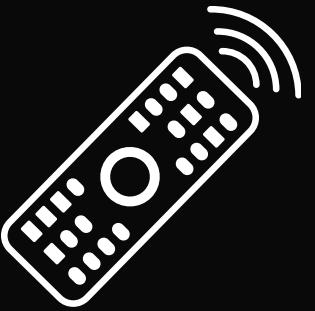
Encapsulamento



Métodos Públicos: Para acessar e modificar os *atributos privados*, são fornecidos métodos públicos (*getters* e *setters*). Esses métodos permitem **acesso** para *recuperar* ou *alterar* os dados.

Exemplo:

```
class Funcionario:  
    def get_id(self):  
        return self._id  
    def set_nome(self, nome):  
        self._nome = nome
```



Instanciamento

Ao **adicionar novos registros** (funcionários, projetos, alocações), o nosso código cria novas instâncias das respectivas classes e adiciona essas instâncias às listas apropriadas.

Exemplo (Adicionar Funcionário):

```
def adicionar_funcionario():
    id = int(input("Digite o ID do funcionário: "))
    nome = input("Digite o nome do funcionário: ")
    cargo = input("Digite o cargo do funcionário: ")
    genero = input("Digite o gênero do funcionário: ")

    funcionario = Funcionario(id, nome, cargo, genero) # Instanciamento
    funcionarios.append(funcionario)
```



Referências

1. Materiais de referência disponibilizados no *Google Classroom* da disciplina “Orientação a Objetos”.
2. Consultas de correções/ajustes do *Chat Copilot Bing* referentes ao código *Python*, relacionando-os ao conteúdo abordado sobre encapsulamento, visibilidade e instanciação.
3. Playlist “Programação Orientada a Objetos - Python”. Disponível em: <<https://youtube.com/playlist?list=PLh4Pa6ArXx2ipvG2roZcwPhHge3VUsrRY&si=sGLzA6SRfYat-wRa>>. Acesso em 09 ago 2024.
4. Vídeo “Como Sair do Zero em Classes no *Python* - Self e Init Explicados”. Disponível em: <https://youtu.be/gomDSZaay3E?si=XaO2S5MbLXSeEL_A> Acesso em 07 ago 2024.
5. Artigo “Entendendo a Programação Orientada a Objetos com *Python*”. Disponível em: <<https://medium.com/@guilhermerdcarvalho/paradigma-orientado-ao-objeto-poo-em-python-a107d35bee3f>> Acesso em 07 ago 2024.

*Agradecemos
a atenção!*

Perguntas?

