# TG1: Streaming de Música

## Programação Orientada à Dados

## Prof. Me. Otávio Parraga


---

## Objetivo

Durante a disciplina discutimos sobre a importância de abstrair o mundo real em objetos e como isso pode facilitar a implementação de sistemas complexos.
O objetivo desta tarefa é aplicar os conceitos de **Programação Orientada a Objetos** para implementar um sistema de **streaming musical simplificado**, inspirado em plataformas como o Spotify.

---

## Descrição

O grupo deve implementar um sistema onde usuários podem criar contas, montar playlists, reproduzir músicas e acompanhar estatísticas de reprodução.

O sistema deve conter um arquivo principal `main.py` responsável por exibir um menu e executar as ações correspondentes.

Todos os **usuários, músicas, artistas e playlists devem ser carregados de um arquivo de configuração em formato Markdown**.
Durante o carregamento, erros devem ser tratados (por exemplo: uma playlist contendo músicas inexistentes). Esses erros devem ser registrados em um arquivo de **log**.

Ao final da execução, o sistema deve gerar:

- um **relatório de reproduções** com todas as analises do sistema (músicas mais ouvidas, usuários mais ativos, etc.)
- e um **arquivo log de erros**.

O desenvolvimento será em grupos de **2 até 3 pessoas**.

---

# Classes Obrigatórias

## Menu

O sistema deve conter um menu com as seguintes opções:

- Entrar como usuário
- Criar novo usuário
- Listar usuários
- Sair

Ao entrar como usuário, o menu deve permitir:

- Reproduzir uma música
- Listar músicas
- Listar podcasts
- Listar playlists
- Reproduzir uma playlist
- Criar nova playlist
- Concatenar playlists
- Gerar relatório
- Sair

---

## Classe Abstrata: ArquivoDeMidia

- **titulo:str**: nome da música, podcast ou álbum.
- **duracao:int**: duração em segundos.
- **artista:str**: artista associado.
- **reproducoes:int**: contador de execuções.

Métodos abstratos:

- **reproduzir()**: simula a execução do arquivo de mídia, mostra na tela as informações contendo título, artista e duração.
- **__eq__()**: compara dois arquivos de mídia (mesmo título e artista).

---

## Música (subclasse de ArquivoDeMidia)

- **genero:str**: (Rock, Pop, Rap, Clássico, etc.)
- **avaliacoes:list[int]**: (lista de notas 0–5)
- **avaliar(nota: int)**: adiciona nota de avaliação.

## Podcast (subclasse de ArquivoDeMidia)

- **episodio:int**: número do episódio.
- **temporada:str**: nome da temporada.
- **host:str**: nome do host.

---

## Playlist

- **nome:str**: nome da playlist.
- **usuario:Usuario**: criador da playlist.
- **itens:list[ArquivoDeMidia]**: lista de ArquivoDeMidia.
- **reproducoes:int**: contador de execuções.

Métodos:

- **adicionar_midia(midia: ArquivoDeMidia)**
- **remover_midia(midia: ArquivoDeMidia)**
- **reproduzir()**: toca todas as mídias da lista. Adiciona um nas reproduções da playlist e um para cada ArquivoDeMidia contido.
- **__add__()**: deve permitir concatenar duas playlists (ex: `playlist1 + playlist2`), resultando em uma terceira playlist que mantenha o nome da primeira, concatene os itens e some as reproduções.
- **__len__()**: retorna o número de itens na playlist.
- **__getitem__()**: permite acessar itens por índice (ex: `playlist[0]`).
- **__eq__()**: compara playlists (mesmo nome, nome das músicas/podcasts e usuário criador).

---

## Usuario

- **qntd_instancias**: contador de usuários criados.
- **nome**: nome do usuário.
- **playlists** (listas criadas pelo usuário)
- **historico** (músicas reproduzidas)

Métodos:

- **ouvir_midia(midia: ArquivoDeMidia)**
- **criar_playlist(nome)**

---

## Analises

Esta classe será a responsável por prover uma série de análises em cima dos objetos criados. A classe deve implementar um conjunto de **métodos estáticos** para gerar as seguintes estatísticas:

- **top_musicas_reproduzidas(musicas: list[Musica], top_n: int) -> list[Musica]**: Retorna as `top_n` músicas mais reproduzidas.
- **playlist_mais_popular(playlists: list[Playlist]) -> Playlist**: Retorna a playlist mais ouvida.
- **usuario_mais_ativo(usuarios: list[Usuario]) -> Usuario**: Retorna o usuário que mais ouviu músicas (usuário com mais músicas no histórico).
- **media_avaliacoes(musicas: list[Musica]) -> dict[str, float]**: Retorna a média de avaliação de cada música.
- **total_reproducoes(usuarios: list[Usuario]) -> int**: Retorna o total de reproduções feitas por todos os usuários.

---

# Tratamento de Erros

O sistema deve lidar com erros, como:

- Música inexistente em uma playlist carregada do arquivo.
- Avaliação inválida (nota fora do intervalo 0–5).
- Usuário já existente ao tentar criar um novo usuário.
- Playlist já existente ao tentar criar uma nova playlist com o mesmo nome para o mesmo usuário.

Os erros devem ser registrados em um arquivo **log**.

---

# Restrições

- Toda classe deve implementar `__str__` e `__repr__`.

- Estatísticas, quando geradas, devem ser salvas em um **relatório**, sendo este um arquivo dentro de uma pasta específica para relatórios.
- Logs de erros devem ser salvos em um arquivo específico para logs, dentro de uma pasta específica para logs.
- O arquivo de configuração deve estar em uma pasta específica para configuração do sistema.
- O histórico do GitHub será avaliado, portanto, **commits devem ser feitos com mensagens claras e descritivas**. Repositórios contendo apenas o código final sem o histórico de desenvolvimento **não serão aceitos**.
- **Todos os membros do grupo devem realizar commits no repositório! Se o aluno não realizar algum commit no repositório, o mesmo terá a nota zerada.**
- Você deverá utilizar Docstrings para documentar suas classes e métodos.
- O código deve ser claro, organizado e seguir boas práticas de programação.
- As bibliotecas permitidas para a implementação do trabalho são: `abs`, `sys`, `os`, `datetime`, `pathlib` e `math`. **Bibliotecas externas não são permitidas,** salvo para a inovação.
- Implementar como **pacote Python** (`Streaming/` com classes, `main.py` fora). Abaixo está um exemplo da estrutura de diretórios a ser seguida:

```
Streaming/
    __init__.py
    menu.py
    arquivo_de_midia.py
    musica.py
    podcast.py
    playlist.py
    usuario.py
    analises.py
main.py
logs/
    erros.log
relatorios/
    relatorio.txt
config/
    dados.md
```

---

# Inovação

O grupo deverá implementar alguma funcionalidade nova no sistema (ex: sistema de recomendação, ranking semanal, compartilhamento de playlists, mix automático, etc.).
A funcionalidade mais votada pelos colegas em sala garantirá **+1 ponto extra**.

---

# Critérios de Avaliação

- Implementação correta das classes (4 pontos)
- Implementação correta das subclasses (1 ponto)
- Sistema de reprodução e playlists (1 ponto)
- Log e relatório (1 ponto)
- **Inovação e criatividade (1 ponto)**
- Apresentação (1 ponto)
- Clareza, documentação e organização (1 ponto)