Universidade do Minho

Licenciatura em Ciências da Computação

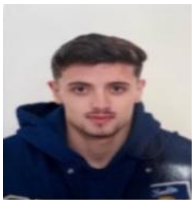
Unidade Curricular de

Programação Orientada a Objetos

Ano Letivo de 2024/2025

SpotifUM

Ivo Sousa *a102935*



Ricardo Cerqueira *a102878*

Miguel Páscoa *a104244*

Índice

Índice de Figuras

1. Introdução

O projeto **SpotifUM** consiste no desenvolvimento de uma aplicação robusta e intuitiva para a gestão e reprodução de conteúdos musicais, inspirada nas principais funcionalidades de plataformas como o Spotify. Desenvolvido em **Java** e alicerçado nos princípios da **Programação Orientada a Objetos (POO)**, o sistema oferece uma arquitetura modular, extensível e fácil de manter.

Entre as suas principais funcionalidades destacam-se:

- **Criação e gestão de utilizadores**, permitindo a personalização da experiência musical de cada indivíduo.
- **Administração de músicas, álbuns e playlists**, com mecanismos intuitivos para organização e categorização de conteúdos.
- **Reprodução de faixas musicais** com suporte a navegação fluida entre playlists, álbuns e recomendações.
- **Geração automática de playlists** com base em critérios definidos ou padrões de comportamento do utilizador.
- **Persistência de dados**, garantindo a salvaguarda do estado da aplicação através de ficheiros, permitindo a continuidade da experiência entre sessões.

Este projeto visa não apenas replicar funcionalidades populares em serviços de streaming, mas também promover a compreensão e aplicação prática de conceitos avançados de POO, como encapsulamento, herança, polimorfismo e design orientado a objetos.

2. Arquitetura de Classes

A arquitetura do sistema foi cuidadosamente concebida para representar de forma clara, coesa e extensível as principais entidades do domínio musical. Seguindo os princípios da **Programação Orientada a Objetos (POO)**, cada classe foi modelada com base em responsabilidades bem definidas, promovendo a reutilização de código e a facilidade de manutenção.

2.1 Classes "model":

- **Utilizador**
  Representa o utilizador da aplicação. Contém atributos como *nome*, *email*, *morada*, o *plano de subscrição*, a *biblioteca pessoal*, as suas *playlists* e o seu *histórico de reproduções*. É a entidade central da interação com o sistema, sendo responsável pela criação de playlists, reprodução de músicas e gestão de conteúdos.

- **Música / MúsicaExplícita**
  Modela uma faixa musical, armazenando metadados como *título*, *artista*, *duração*, *notação*, *género* e *letra*. A subclasse **MúsicaExplícita** especializa a classe base para representar conteúdos com linguagem imprópria ou restrições

etárias, permitindo uma gestão diferenciada de acordo com o tipo de utilizador ou plano.

- **Álbum**
  Agrega um conjunto de músicas relacionadas, tipicamente organizadas por um mesmo artista. Cada álbum possui um *título*, *artista associado* e uma *lista de faixas*, refletindo a estrutura comum de lançamentos musicais no mercado.

- **Playlist**
  Representa uma lista de reprodução personalizada. Pode ser criada **manualmente** pelo utilizador ou **gerada automaticamente** pelo sistema com base em critérios específicos. Possui características adicionais como *visibilidade pública/privada* e *modo de reprodução aleatório*.

- **Biblioteca**
  A biblioteca agrega todos os **álbuns** e **playlists pessoais ou externas** que o utilizador escolheu guardar, funcionando como uma coleção musical pessoal persistente ao longo do tempo. Sendo que os utilizadores do plano Free, não podem guardar qualquer tipo de álbum ou playlist.

- **PlanoSubscricao** (e subclasses **Free**, **PremiumBase**, **PremiumTop**)
  Define o nível de acesso e os privilégios do utilizador. Cada plano implementa políticas distintas de:

  1) Acesso a funcionalidades premium
  2) Geração automática de playlists
  3) Atribuição de **pontos** por reprodução

2.2 Classe “utils”:

- **Serializer:** Classe utilitária responsável por realizar a **persistência de dados** da aplicação, permitindo **guardar** objetos Java em ficheiros através de serialização e, posteriormente, **carregá-los** de volta para memória, preservando o estado da aplicação entre execuções.

2.3 Classe “controller”:

- **SpotifumController:** Classe responsável por centralizar e coordenar toda a **lógica funcional da aplicação**, atuando como intermediário entre os dados do sistema

(model) e a interface com o utilizador (view). Gere operações sobre utilizadores, músicas, playlists, álbuns e estatísticas, garantindo o correto funcionamento das regras de negócio.

2.4 Classe “view”:

- **Menu:** Classe que implementa a **interface textual da aplicação**, permitindo ao utilizador interagir com o sistema através da consola. Apresenta menus contextuais consoante o tipo de utilizador e encaminha as suas ações para o controlador, assegurando uma experiência de navegação intuitiva.

3. Funcionalidades Desenvolvidas

3.1 Criação de utilizadores com planos distintos

A aplicação permite a criação de novos utilizadores com associação a um dos três planos de subscrição disponíveis: **Plano Free**, **Plano PremiumBase** e **Plano PremiumTop**. Cada plano confere ao utilizador diferentes permissões e vantagens, influenciando diretamente a sua experiência na aplicação.

3.2 Leitura inicial de dados a partir de ficheiro

No arranque da aplicação, é possível carregar dados previamente definidos (músicas, álbuns, playlists) a partir de um ficheiro de texto. Esta funcionalidade permite iniciar o sistema com conteúdo pré-carregado e facilita a integração com fontes externas de dados.

3.3 Prevenção de criação de utilizadores com email duplicado ou inválido

Foi implementada uma verificação rigorosa durante o processo de registo de utilizadores, impedindo a criação de contas com emails **vazios** ou **duplicados**. Esta validação garante unicidade no sistema e evita inconsistências nos dados.

3.4 Criação de músicas

A aplicação permite criar instâncias de músicas completas, com atributos como o nome, intérprete, editora, letra, acordes musicais, género e duração em segundos. Suporta também músicas explícitas, que estendem a classe Musica.

3.5 Criação e reprodução de álbuns

Os utilizadores podem criar álbuns personalizados, agrupando músicas existentes sob um título e artista principal. Além disso, é possível reproduzir qualquer álbum, tanto da própria biblioteca como do catálogo global, respeitando a ordem definida das faixas.

3.6 Criação e reprodução de playlists (manuais e automáticas)

É possível criar playlists de forma **manual**, selecionando músicas do catálogo, ou gerar playlists **automáticas** com base em critérios como género musical, músicas favoritas ou explícitas. As playlists podem ser públicas ou privadas e configuradas para reprodução aleatória ou sequencial.

3.7 Reprodução aleatória para utilizadores Free

Utilizadores com plano Free têm acesso limitado ao sistema, podendo apenas reproduzir músicas de forma **aleatória** a partir do catálogo geral. Esta limitação simula um modelo gratuito com menos controlo sobre os conteúdos.

3.8 Reprodução com controlo (avançar, pausar, recuar) para Premium's

Utilizadores Premium têm acesso a um sistema de reprodução com controlo total: podem **avançar**, **recuar** e **pausar** músicas durante a reprodução de playlists ou álbuns. Esta funcionalidade melhora a experiência de navegação e permite maior liberdade de escolha.

3.9 Atribuição de pontos por reprodução, diferenciada por plano

O sistema atribui pontos aos utilizadores com base no seu plano de subscrição. O plano Free oferece poucos ou nenhuns pontos, o PremiumBase atribui pontos fixos por reprodução, e o PremiumTop atribui uma percentagem dos pontos acumulados **apenas quando uma música nova é ouvida**.

3.10 Estatísticas

A aplicação compila e apresenta várias estatísticas globais como: a música mais reproduzida, o intérprete mais escutado, o género dominante, o utilizador com mais pontos, e o mais ativo. Estas métricas são úteis para análise de uso e valorização do conteúdo.

3.11 Persistência do estado da aplicação (utilizadores, músicas, álbuns, playlists)

Todo o estado da aplicação é guardado de forma persistente utilizando serialização. Isto garante que, após encerrar o programa, os dados criados pelos utilizadores são mantidos e carregados na execução seguinte.

3.12 Leitura e escrita de ficheiros de estado usando Serializer.java

A classe Serializer encapsula a lógica de leitura e escrita de objetos Java em ficheiros .dat, através da API de serialização. Foi usada para guardar separadamente os utilizadores, músicas, álbuns e playlists, garantindo modularidade e recuperação eficiente dos dados.

4. Diagrama de Classes

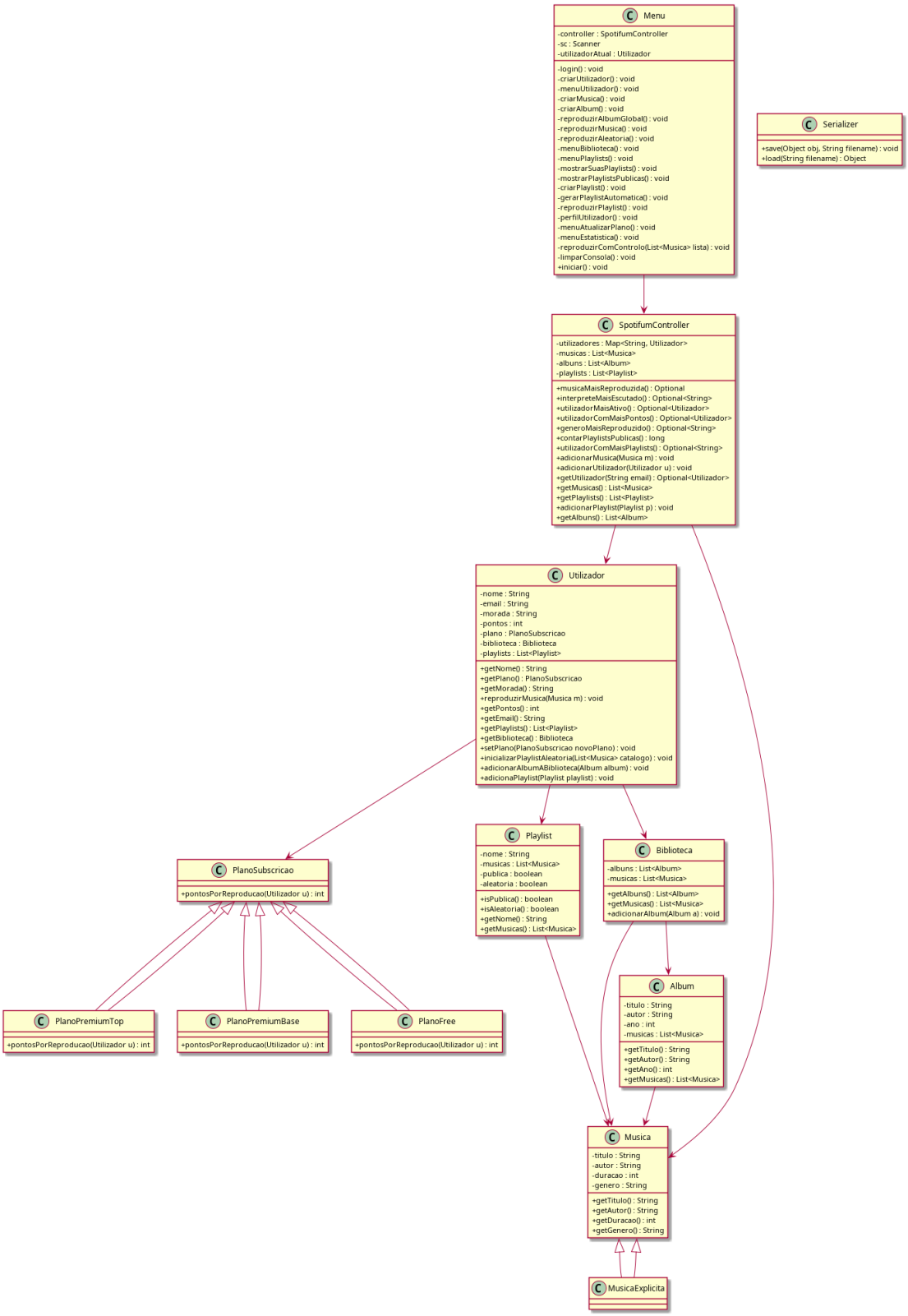


Figura 1 - Diagrama de Classes

5. Considerações Finais

O projeto **SpotifUM** foi desenvolvido respeitando integralmente os requisitos estabelecidos no enunciado da unidade curricular de Programação Orientada aos Objetos. A aplicação apresenta uma estrutura robusta, baseada nos princípios fundamentais da programação orientada a objetos, nomeadamente o **encapsulamento**, a **modularidade**, e a **abstração** de responsabilidades.

A interação com o utilizador é assegurada através de uma **interface textual intuitiva**, que adapta as opções disponíveis conforme o tipo de plano de subscrição. O sistema permite uma navegação fluida entre funcionalidades essenciais e avançadas, como a criação e reprodução de conteúdos musicais, gestão de biblioteca, estatísticas globais e playlists geradas automaticamente.

Foi implementada uma **validação rigorosa dos dados introduzidos**, evitando, por exemplo, a criação de utilizadores com emails vazios ou repetidos, contribuindo para a integridade e fiabilidade da aplicação.

A separação clara entre as camadas de **modelo (dados e lógica de negócio)**, **vista (interface)** e **controlo (orquestração de operações)** garante um design limpo, de fácil manutenção e escalável.

Por fim, destaca-se a implementação de **mecanismos de persistência** que asseguram a salvaguarda do estado da aplicação entre sessões, permitindo aos utilizadores retomar a sua experiência sem perda de dados, mesmo após o encerramento da aplicação.