Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Programação Orientada aos Objetos

Ano Letivo de 2023/2024

SpotifUM

Afonso Martins
a106931

Luis Felício
a106913

Goncalo Castro
a107337

17 de maio de 2025

POO

Data da Receção	
Responsável	
Avaliação	
Observações	

SpotifUM

Afonso Martins a106931
Luis Felício a106913
Goncalo Castro a107337

17 de maio de 2025

Resumo

O presente trabalho teve como objetivo desenvolver uma aplicação de gestão de músicas e playlists para utilizadores de um serviço de streaming musical denominado SpotifUM. A aplicação permite registar diferentes tipos de conteúdos musicais, incluindo músicas, álbuns e playlists, além dos dados dos utilizadores e seus perfis de subscrição (Free, PremiumBase e PremiumTop).

Foram implementadas funcionalidades para criar e gerir utilizadores, músicas e playlists. É possível registar a reprodução de músicas pelos utilizadores, calculando automaticamente os pontos atribuídos com base no plano de subscrição do utilizador. Adicionalmente, foram desenvolvidos mecanismos para criar diferentes tipos de playlists, incluindo playlists aleatórias, playlists definidas pelo utilizador e listas de favoritos geradas a partir das preferências do utilizador.

O programa permite ainda gerar playlists personalizadas para cada utilizador premium, de acordo com critérios como preferências musicais, tempo máximo de duração e tipos específicos de música, como as músicas explícitas. Foram implementadas funcionalidades para obter várias estatísticas, como a música mais reproduzida, o intérprete mais escutado, o utilizador com mais pontos, entre outras.

A aplicação encontra-se totalmente funcional, permitindo a gestão completa de utilizadores, músicas e playlists, assim como a obtenção de estatísticas relevantes. Todos os requisitos propostos foram cumpridos, adotando-se boas práticas de programação orientada a objetos, além de garantir a robustez e tratamento adequado de erros. A aplicação é capaz de armazenar e recuperar o seu estado num ficheiro, facilitando a sua utilização contínua.

Área de Aplicação: Desenvolvimento no âmbito da Programação Orientada aos Objetos.

Palavras-Chave: Streaming musical, gestão de conteúdos, arquitetura de aplicação, JAVA, design pattern, MVC, controlo de qualidade de código.

Índice

Lista de Figuras

1. Introdução

Este relatório apresenta o trabalho desenvolvido no âmbito da Unidade Curricular de Programação Orientada a Objetos (POO), no ano letivo de 2024/2025. O objetivo do projeto consistiu na criação de uma aplicação interativa — designada **SpotifUM** — destinada à gestão de conteúdos musicais e perfis de utilizadores, com funcionalidades que vão desde a reprodução de músicas à criação e gestão de playlists e álbuns personalizados.

Ao longo do desenvolvimento, o grupo enfrentou vários desafios técnicos e conceptuais, sendo necessário aplicar os princípios da programação orientada a objetos para conceber uma arquitetura sólida, modular e escalável. Foram tomadas decisões estruturais com base nos requisitos progressivos definidos no enunciado, desde funcionalidades elementares de gestão de músicas e utilizadores, até à implementação de subscrições premium com benefícios distintos, playlists personalizadas geradas automaticamente e estatísticas de utilização.

1.1. Organização de requisitos

Inicialmente, foi realizada uma análise detalhada dos requisitos especificados no enunciado do projeto, permitindo identificar as entidades principais e suas interações:

- **Músicas**: Representam o conteúdo base da aplicação, com informações como nome, intérprete, editora, género, letra, duração, e eventualmente classificações especiais como **MusicaExplicita** ou **Multimédia**.
- **Utilizadores**: Divididos em categorias como **Free**, **PremiumBase** e **PremiumTop**, cada uma com permissões e funcionalidades distintas. Os utilizadores podem acumular pontos, criar playlists e guardar álbuns dependendo do plano.
- **Playlists**: Estruturas que agrupam músicas. Podem ser criadas manualmente por utilizadores premium, geradas automaticamente com base nos hábitos de reprodução ou construídas com critérios específicos como género musical e tempo máximo.

Com base nestas entidades, os requisitos foram agrupados nas seguintes categorias:

1. **Gestão de entidades:**
   - Criação de utilizadores, músicas e playlists.
   - Associação de utilizadores a planos de subscrição.
   - Recolha de estatísticas sobre reproduções e preferências.
2. **Reprodução de conteúdos:**
   - Execução de músicas e playlists, com diferentes comportamentos consoante o plano do utilizador.
   - Simulação da reprodução através da apresentação da letra da música.
3. **Pontuação e estatísticas:**
   - Atribuição de pontos por reprodução conforme o plano.
   - Cálculo de indicadores como músicas mais reproduzidas, utilizadores mais ativos, playlists mais comuns, entre outros.
4. **Registo detalhado de utilizadores:**

- Manter registos do código do utilizador, nome, email,morada , pontos, musicasFavoritas, plano e reproducoes.
- Associar Musicas ouvidas aos utilizadores, com o respetivo tempo de quando foi ouvida.

5. **Funcionalidades avançadas:**
   - Geração de playlists automáticas com base no histórico de reprodução do utilizador.
   - Filtros como género musical, tempo máximo ou apenas músicas explícitas.

1.2. Estratégia de desenvolvimento

A implementação foi realizada em **Java**, respeitando boas práticas como convenções de nomeação, modularização e documentação com **Javadoc**. O sistema de construção escolhido foi o **Gradle 8.6**.

Para os testes, foi utilizada a biblioteca **JUnit 5**, que permitiu criar testes unitários eficazes e automatizados, garantindo maior fiabilidade no comportamento das funcionalidades desenvolvidas.

1.3. Design patterns

Durante o projeto, foram aplicados vários padrões de design para garantir uma estrutura sólida e extensível:

- **Abstract Factory**: Facilitou a criação de objetos para diferentes tipos de utilizadores e músicas, mantendo uma interface comum e permitindo uma expansão flexível da aplicação.
- **Facade**: Centralizou a lógica de interação com o sistema, simplificando o acesso às funcionalidades principais e promovendo o encapsulamento da complexidade interna.

1.4. Estratégia de desenvolvimento

O grupo utilizou a plataforma Discord como ferramenta para planear, acompanhar e distribuir as tarefas entre os três membros do grupo, permitindo o acompanhamento do progresso de cada membro do grupo.

Em relação ao código, como observável no bloco de código em baixo, optámos por um modelo padrão de programação em Java, aplicando boas práticas de produção de código, como convenções de nomenclatura, organização lógica e modularização de código. Isso assegura clareza e consistência, facilita a manutenção e maximiza a legibilidade. Também adotámos o padrão Javadoc para documentar as classes, métodos e parâmetros, oferecendo uma referência clara e estruturada para futuros desenvolvedores. Como ferramenta de compilação utilizamos Gradle 8.6.

```java
/**
 * Verifica se uma música está nos favoritos do Utilizador
 *
 * @param email Email do Utilizador
 * @param musicName Nome da música
 * @return true se a música estiver nos favoritos, false caso contrário
 */
public boolean isMusicFavorite(String email, String musicName) {
    List<Musica> favorites = getFavoriteMusics(email);
    if (favorites == null) {
        return false;
    }

    return favorites.stream()
            .anyMatch(m -> m.getNome().equalsIgnoreCase(musicName));
}
```

Para a realização de testes, recorremos à biblioteca JUnit 5. Esta biblioteca fornece ferramentas modernas e flexíveis para a criação de testes unitários, permitindo a definição de conjuntos de testes com verificações precisas para verificar o comportamento esperado das funcionalidades. Além disso, a JUnit 5 suporta a execução automatizada de testes, possibilitando a identificação precoce de problemas durante o desenvolvimento. Segue um exemplo descritivo de testes de inserção de um plano de treino:

```java
  /**
 * Tests the generation of a music playlist for a Premium user.
 * Verifies that only the favorite genres of the user are included in the playlist.
 */
@Test
void testGeneratedMusicsForPremiumUser() {
    Utilizador premiumUser = userManager.createUser("Premium User",
"premium@example.com", "password123", PlanoSubscricao.PremiumTop);

    premiumUser.addMusicToFavorites(musicas.get(0)); // Song1 (POP)
    premiumUser.addMusicToFavorites(musicas.get(1)); // Song2 (ROCK)
    //List<GeneroMusical> preferredGenres = premiumUser.getPreferredGenres();

    GeneratedMusics generatedPlaylist = new GeneratedMusics(musicas, premiumUser, 0,
false);

    List<Musica> playlistMusics = generatedPlaylist.getMusicas();
    assertEquals(2, playlistMusics.size()); // POP e ROCK
    assertEquals("Song1", playlistMusics.get(0).getNome());
    assertEquals("Song2", playlistMusics.get(1).getNome());
}
```

Na aplicação, pretendemos implementar a estratégia de agregação como parte do encapsulamento previsto em Programação Orientada aos Objetos (POO). A agregação é uma relação entre objetos em que um objeto contém ou possui referências a outros objetos, mas a vida desses objetos agregados não depende do objeto que os contém.

Esta estratégia permite criar estruturas mais complexas a partir de objetos mais simples, promovendo a reutilização de código e a flexibilidade no design do sistema. Ao encapsular funcionalidades específicas em diferentes classes, agregamos essas classes em entidades maiores para oferecer funcionalidades mais amplas.

Por exemplo, ao modelar componentes importantes da aplicação, como utilizadores, utilizamos a agregação para reunir características e comportamentos distintos em classes separadas. Em seguida, essas classes foram agregadas numa classe maior que as administra e as coordena.

A aplicação desta estratégia não só promove o encapsulamento, permitindo que cada classe se concentre nas suas responsabilidades específicas, como também facilita a manutenção e a evolução do código. Isto porque as mudanças podiam ser feitas de forma isolada, sem impactar outras partes da aplicação.

1.5. Design patterns

O grupo avançou com a projeção da aplicação ao estudar design patterns para identificar quais os padrões de desenvolvimento mais adequados aos requisitos. Um padrão que se destacou foi o **Abstract Factory**, que permite a criação de famílias de objetos relacionados ou dependentes sem definir explicitamente suas classes concretas. No contexto das atividades e dos utilizadores, este padrão é crucial para garantir a implementação correta de famílias de atividades e as suas interações, como ilustrado na Figura 1 e na Figura 2. O **Abstract Factory** possibilita a criação de objetos específicos para diferentes tipos de atividades, mantendo uma interface consistente e permitindo a extensão fácil do sistema para novos tipos de atividades e utilizadores.

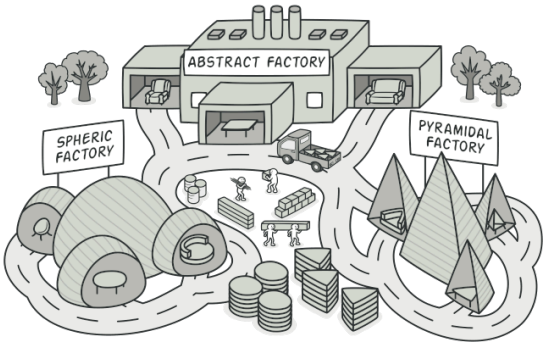


Figura 1: Exemplo de representação gráfica do padrão Abstract Factory. [1]

Figura 2: Exemplo de representação concetual do padrão Abstract Factory na aplicação.

Durante a investigação sobre padrões de design, identificámos o padrão **Facade** como especialmente relevante. Este padrão permite criar uma única interface de acesso para um conjunto de funcionalidades relacionadas, centralizando as operações e ocultando a complexidade interna das classes subjacentes. Ao encapsular as interações com subsistemas, o **Facade** simplifica a utilização da aplicação e garante uma maior modularidade e coesão, ao mesmo tempo em que previne o acesso direto ao conhecimento interno das classes, proporcionando assim uma camada de abstração adicional para o utilizador. Observe-se na Figura 3 e Figura 4 as representações gráfica e concetual, respetivamente.

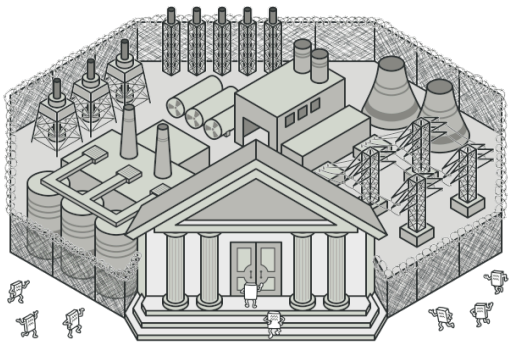


Figura 3: Exemplo de representação gráfica do padrão Facade. [2]

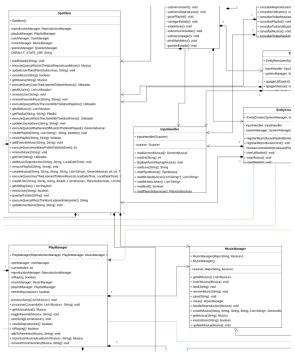


Figura 4: Exemplo de representação concetual do padrão Facade na aplicação.

2. Diagrama de classes

Nesta secção do relatório, será apresentada a construção do sistema com base no diagrama de classes. A ferramenta “draw.io” foi utilizada para modelar o sistema, empregando a notação UML (Unified Modeling Language) para a representação gráfica das classes e as suas relações.

O diagrama de classes é uma parte crucial do design do sistema, pois oferece uma visão estrutural dos componentes da aplicação. Descreve as classes que compõem o sistema, os seus atributos e métodos, além de ilustrar as relações entre as classes, como herança, associação e dependência.

Ao apresentar o diagrama de classes, esta secção fornecerá uma visão abrangente da estrutura do sistema, destacando as classes principais, as suas responsabilidades, atributos e métodos, bem como as interações entre elas. Isso garantirá uma base sólida para entender a arquitetura do sistema e como ele foi projetado para atender aos requisitos estabelecidos inicialmente.

2.1. Modelação das Playlists

Começou-se por modelar as playlists de acordo com o padrão Abstract Factory anteriormente apresentado. Nesse contexto, criou-se uma interface denominada “PlaylistInterface”, que obriga à implementação de métodos essenciais ao bom funcionamento da aplicação.

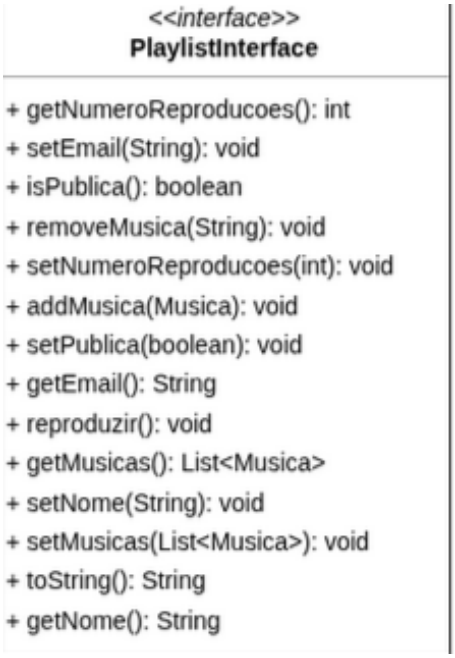


Figura 5: Representação da interface “PlaylistInterface” no diagrama de classes.

A classe `Playlist` representa uma coleção de músicas pertencente a um determinado utilizador no sistema. Esta entidade é central para a gestão de conteúdos musicais, permitindo organizar músicas, controlar a sua visibilidade (pública ou privada), e acompanhar estatísticas como o número de reproduções.

A `Playlist` contém informação relevante como o nome, o email do utilizador que a criou, a lista de músicas que a compõem e se a mesma é pública ou não. Para além disso, fornece métodos para manipulação da lista de músicas, consulta e modificação dos seus atributos, reprodução e clonagem da playlist

Figura 6: Representação da classe concreta “Playlist” no diagrama de classes.

Esta classe possui igualmente duas subclasses, a “GeneratedMusics” e a “FavoriteMusics” que são classes para para representar as playlists geradas automaticamente e as músicas favoritas de um dado utilizador, respetivamente.

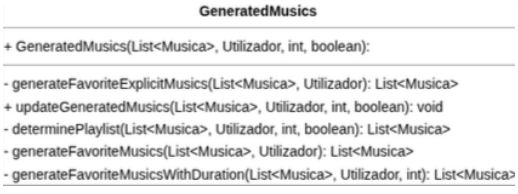


Figura 7: Representação da subclasse “GeneratedMusics” no diagrama de classes.

Figura 8: Representação da subclasse “FavoriteMusics” no diagrama de classes.

Além disso a aplicação possui, um PlaylistManager que permite administrar as playlist da aplicação e possui também um *map* para armazenar as playlist e facilitar a gestão e a recuperação das mesmas.

Figura 9: Representação da classe concreta “PlaylistManager” no diagrama de classes.

Também ligada a esta modelação, assim como a outras, esta a classe PlayManager que é responsável por gerir a reprodução de músicas na aplicação. Centraliza a lógica de reprodução, permitindo alternar entre músicas, iniciar/parar reprodução e gerir favoritas. Para isso, interage com os gestores de utilizadores, músicas, playlists e reproduções. Possui métodos para navegar entre músicas, reproduzir a atual, processar comandos do utilizador e atualizar favoritos. É essencial na ligação entre os dados do sistema e a experiência de reprodução.

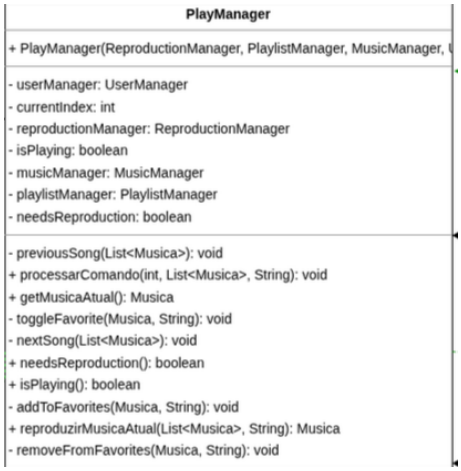


Figura 10: Representação da classe concreta “PlayManager” no diagrama de classes.

2.2. Modelação dos Utilizadores

Na modelação dos utilizadores, foi adotado o padrão Abstract Factory, criando uma estrutura consistente para a gestão de diferentes tipos de utilizadores na aplicação. O ponto de partida foi a definição da interface “UtilizadorInterface”, que impõe a implementação de métodos essenciais para o correto funcionamento da aplicação.

Figura 11: Representação da interface "UtilizadorInterface" no diagrama de classes.

Em seguida criou-se a classe Utilizador que representa os utilizadores da aplicação, sendo responsável por armazenar as suas informações e disponibilizar os métodos necessários para a sua gestão e interação com o sistema. Cada utilizador possui campos como nome, endereço de email e plano de subscrição, que define o tipo de acesso ou funcionalidades disponíveis na aplicação.

Figura 12: Representação da classe concreta "Utilizador" no diagrama de classes.

O *field* plano é definido e calculado a partir do *enum* que possui o seguinte diagrama de classes:

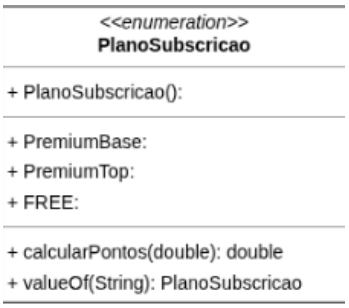


Figura 13: Representação do enumerável “PlanoSubscricao” no diagrama de classes.

Além disso, a aplicação possui um gestor de utilizadores, denominado “UtilizadorManager”. Este gestor é responsável pela administração de todos os utilizadores na aplicação, mantendo um *Map* que armazena os diferentes utilizadores registados, permitindo a fácil recuperação e gestão dos mesmos.

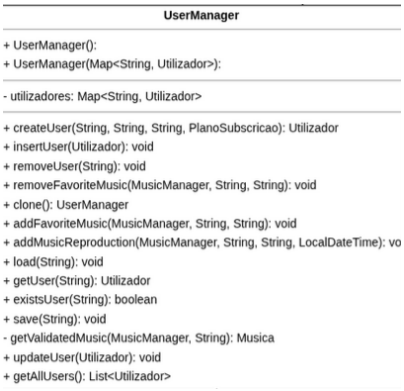


Figura 14: Representação da classe concreta “UtilizadorManager” no diagrama de classes.

2.3. Modelação das Músicas

A modelação das músicas, foi começada pela implementação da interface das músicas intitulada “MusicaInterface”. Esta é responsável pela correta implementação dos métodos na aplicação.

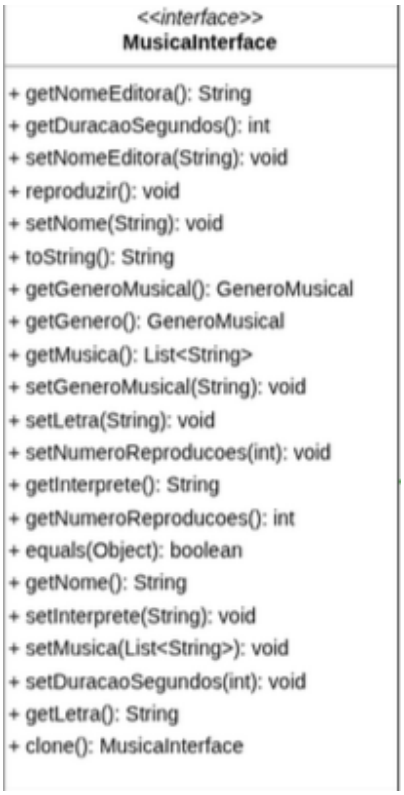


Figura 15: Representação da interface “MusicaInterface” no diagrama de classes.

A classe Musica foi implementada como a classe base que representa o conceito genérico de uma música. Esta classe contém os atributos e comportamentos comuns a todas as músicas, como o título, artista, duração, o tipo de música, entre outros.

Figura 16: Representação da classe concreta “Musica” no diagrama de classes.

Esta classe possui um campo “GeneroMusical” que é definido neste diagrama.

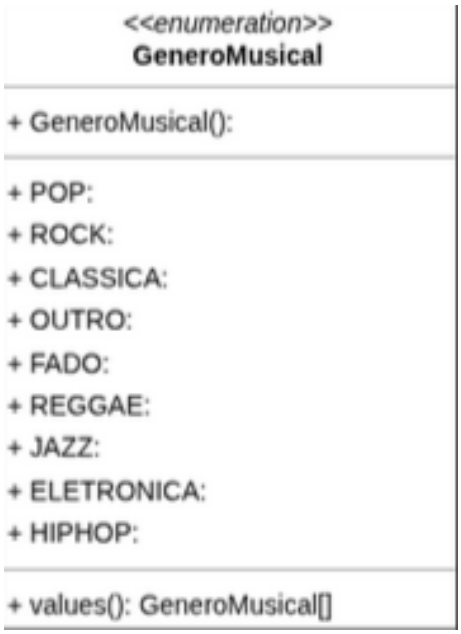


Figura 17: Representação do enumerável “GeneroMusical” no diagrama de classes.

Para além desta, possui duas subclasses, a Multimedia que implementa uma musica com vídeo e a Explicita que é uma música com letra explícita.

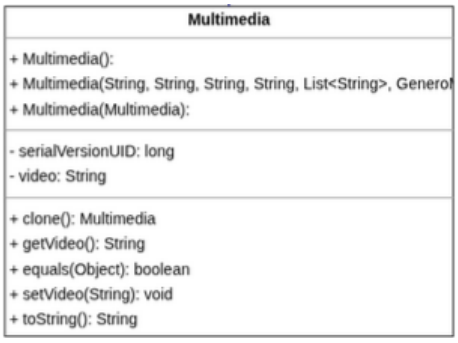


Figura 18: Representação da subclasse “Multimedia” no diagrama de classes.

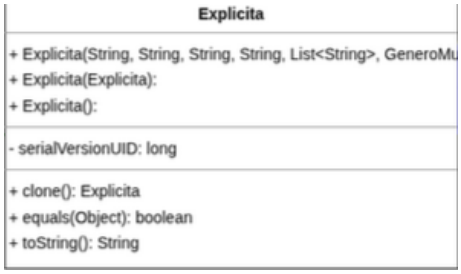


Figura 19: Representação da subclasse “Explicita” no diagrama de classes.

Além disso, a aplicação deverá incluir um gestor específico para músicas, denominado “MusicManager”. Este gestor é responsável pela criação, armazenamento e gestão das músicas na aplicação. Mantém um *Map* que armazena as diferentes músicas.

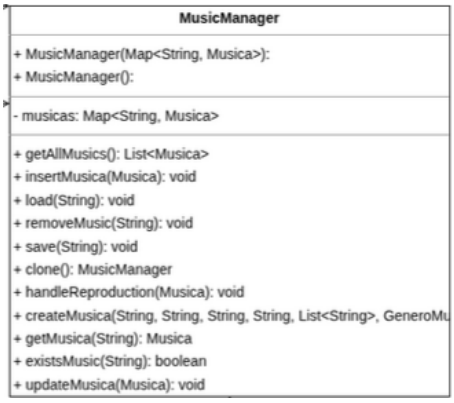


Figura 20: Representação da classe concreta “MusicManager” no diagrama de classes.

2.4. Modelação das Reproduções Musicais

A implementação da modelação das Reproduções musicais, provém da necessidade de aceder às reproduções de cada utilizador e de as ter armazenadas de forma a implementar as *queries*. Começámos pela criação da interface chamada "ReproducaoMusicalInterface" que assegura o bom funcionamento e chamada dos métodos implementados.

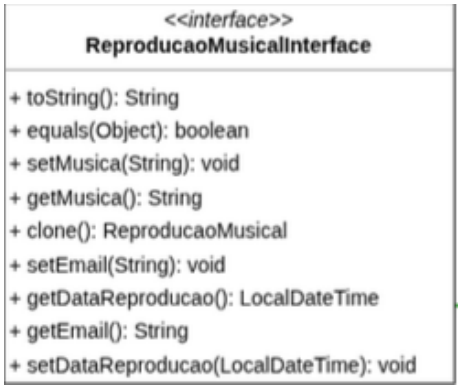


Figura 21: Representação da interface "ReproducaoMusicalInterface" no diagrama de classes.

Após esta interface criámos a classe "ReproducaoMusical" que tem os métodos necessários para criar e gerir as reproduções. Estas possuem o nome da música, o email do utilizador responsável pela reprodução e a data e hora (*LocalDateTime*) em que foi reproduzida.

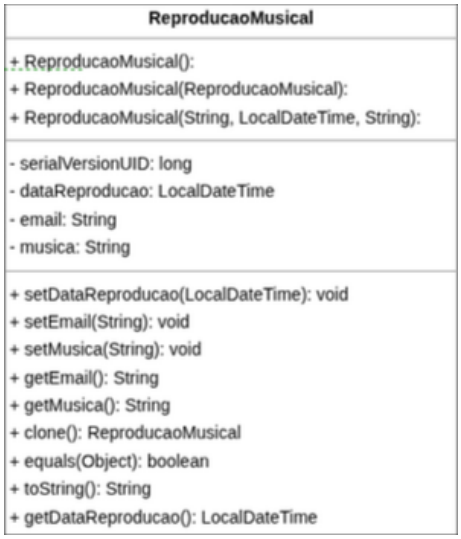


Figura 22: Representação da classe concreta "ReproducaoMusicalInterface" no diagrama de classes.

Para gerir e armazenar todas as reproduções musicais foi implementado um *manager* que possui um *map* que guarda todas as reproduções e permite um rápido acesso a cada uma delas. Este *manager* intitula-se "ReproductionManager". Este possui métodos para lidar com a reprodução de músicas.

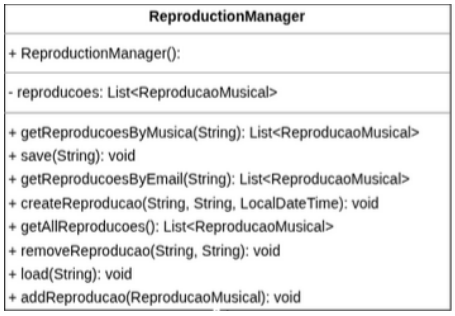


Figura 23: Representação da classe concreta "ReproducaoManager" no diagrama de classes.

2.5. Modelação das classes de utilitários

2.5.1. Menu

A classe `Menu` é responsável por gerir a interface de interação com o utilizador na aplicação **Spotifum**. Controla o ciclo principal do programa, apresentando as opções do menu e encaminhando o utilizador para as diferentes operações disponíveis.

Possui os seguintes campos principais:

- `spotifum`: núcleo da aplicação que gere os dados do sistema;
- `inputHandler`: trata das entradas do utilizador;
- `entityCreator`: permite criar novas entidades (utilizadores, músicas, playlists);
- `entityViewer`: permite consultar informações;
- `entityRemover`: permite remover entidades existentes.

A sua estrutura modular promove organização e facilita a manutenção do código.

Figura 24: Representação da classe concreta “Menu” no diagrama de classes.

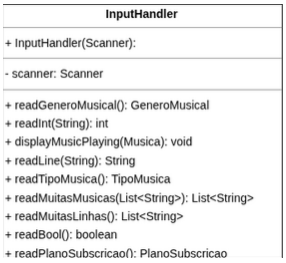


Figura 25: Representação da classe concreta “InputHandler” no diagrama de classes.

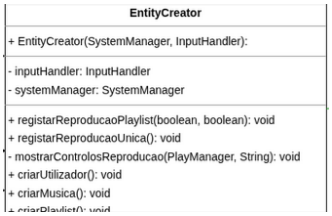


Figura 26: Representação da classe concreta “EntityCreator” no diagrama de classes.

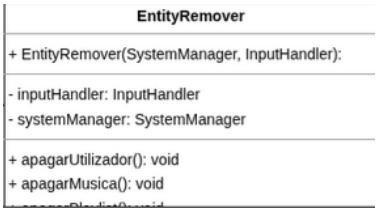


Figura 27: Representação da classe concreta “EntityRemover” no diagrama de classes.

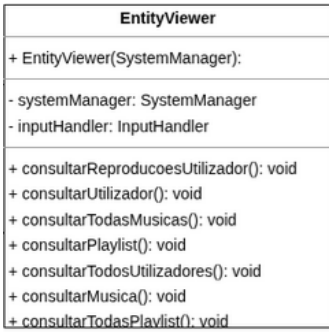


Figura 28: Representação da classe concreta “EntityViewer” no diagrama de classes.

2.5.2. Estatísticas

Construção de um gestor para as estatísticas pedidas no enunciado do projeto.

Figura 29: Representação da classe concreta “QueriesManager” no diagrama de classes.

2.5.3. Exceções

Com o seguimento do projeto surgiu o objetivo de especificar algumas exceções já presentes na tecnologia e, como tal, dedicámos algumas classes à sua definição e implementação.

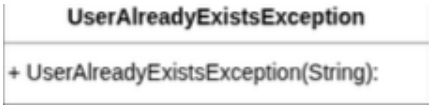


Figura 30: Representação da classe concreta “UserAlreadyExistsException” no diagrama de classes.

Figura 31: Representação da classe concreta “UserNotFoundException” no diagrama de classes.

Figura 32: Representação da classe concreta “MusicNotFoundException” no diagrama de classes.

2.6. Modelação do MVC

A aplicação foi desenvolvida segundo uma abordagem inspirada no padrão arquitetural MVC (**Model-View-Controller**), apesar de não ter sido necessária a implementação explícita de um **Controller**, uma vez que não existem interações complexas que exijam mediação entre o modelo de dados e a interface.

Neste modelo simplificado:

- O **Model** é representado pela classe `Spotifum`, que agrega toda a lógica de negócio e manipulação dos dados da aplicação. Esta classe coordena os diferentes gestores internos (como `ReproductionManager`, `PlaylistManager`, `UserManager`, entre outros), sendo responsável por operações como a criação e remoção de entidades, consultas estatísticas e persistência do estado.

Figura 33: Representação da classe concreta “Spotifum” no diagrama de classes.

- A **View** é representada pela classe `Menu`, que fornece toda a interface textual com o utilizador. Esta classe é responsável por apresentar menus, receber entradas e invocar os métodos do `Spotifum` conforme as ações do utilizador. A `Menu` delega tarefas específicas a outras componentes auxiliares como `InputHandler`, `EntityViewer`, `EntityRemover`, `SystemManager` e `EntityCreator`, que ajudam na interação com os dados.

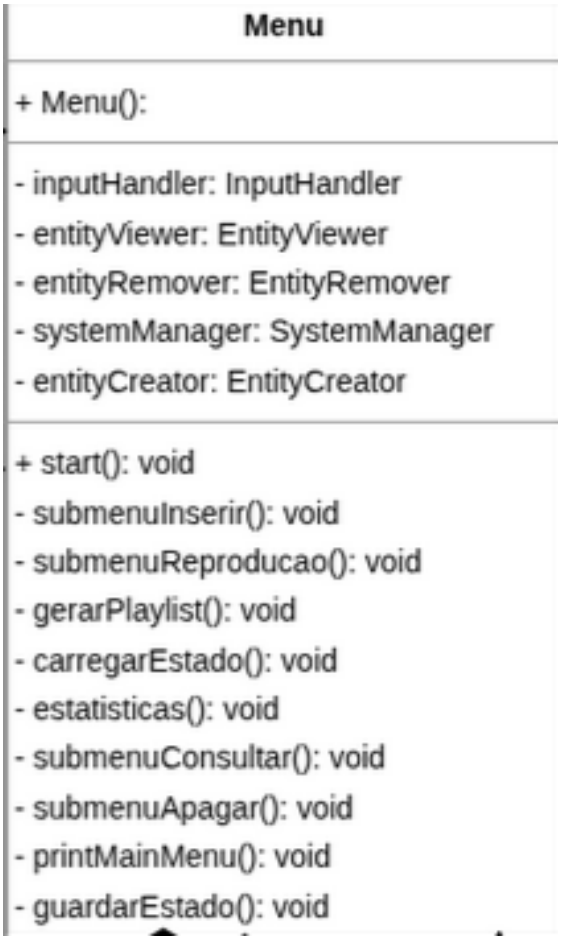


Figura 34: Representação da classe concreta “Menu” no diagrama de classes.

- A classe `Main` apenas inicia a aplicação, instanciando o `Menu` e chamando o seu método `start`.

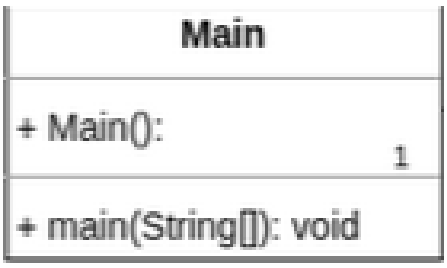


Figura 35: Representação da classe concreta “Main” no diagrama de classes.

Este design permite uma separação clara entre a lógica do programa e a interface com o utilizador, promovendo modularidade e facilitando a manutenção e evolução do sistema.

2.7. Modelação final

Finalmente, foram estabelecidas todas as relações entre as várias representações apresentadas, algumas que, com o avançar do projeto, sofreram pequenas alterações. Como o diagrama de classes final não é

apresentável no espaço reservado para este ficheiro, está disponibilizado como anexo externo na mesma diretoria do relatório atual.

3. Implementação

Nesta secção do relatório, será apresentada a implementação, onde é representado o culminar de esforços meticulosos e de um planeamento estratégico. Este relatório oferece uma visão detalhada dos processos, estratégias e resultados alcançados durante a fase de execução. Vamos explorar os objetivos definidos, as soluções desenvolvidas para ultrapassar desafios, proporcionando uma avaliação abrangente do sucesso da implementação.

3.1. Contextos base da aplicação

A arquitetura da aplicação **SpotifUM** foi construída com base em princípios de modularidade e extensibilidade, apoiando-se no padrão de design **Abstract Factory** para gerir a criação de diferentes tipos de utilizadores e músicas. Esta abordagem permitiu manter uma interface comum para as entidades centrais da aplicação, ao mesmo tempo que garantiu a flexibilidade necessária para acomodar diferentes comportamentos e características.

Para promover a clareza estrutural e a separação de responsabilidades, o sistema foi organizado em **packages** distintos, de acordo com os domínios principais:

- **Músicas**: As músicas constituem o núcleo de conteúdos da aplicação. Foram agrupadas num **package** próprio, contendo classes para os diferentes tipos, como músicas explícitas (**Explicita**) ou com componentes multimédia (**Multimedia**). Cada subclasse implementa comportamentos específicos, como a exibição da letra, reprodução adaptada ou informações adicionais, respeitando o princípio da especialização.
- **Utilizadores**: Os utilizadores estão organizados num **package** dedicado, onde está defenidos os plano como: **Free**, **PremiumBase** e **PremiumTop** e a interface. Cada tipo define permissões e funcionalidades específicas, como criação de playlists, acesso a funcionalidades premium ou acumulação de pontos. Esta estrutura facilita a aplicação de restrições ou benefícios exclusivos com base no plano, e permite a expansão futura com novos tipos de subscrição.
- **Playlists**: As playlists são agrupamentos de músicas personalizáveis consoante o plano do user , criadas manualmente pelos utilizadores ou geradas automaticamente com base no seu histórico de reprodução. Foram implementadas num **package** independente, com suporte para critérios como género musical, tempo máximo e exclusividade de músicas explícitas.
- **Gestor Central (Facade)**: A camada de fachada centraliza o acesso às funcionalidades principais da aplicação — como reprodução de músicas, criação de playlists ou consulta de estatísticas — ocultando a complexidade das interações internas. Este padrão promove a modularidade, melhora a legibilidade do código e reduz o acoplamento entre componentes.

  Esta organização permitiu um desenvolvimento mais limpo, escalável e aderente aos princípios da programação orientada a objetos, assegurando que cada componente da aplicação cumpre um papel bem definido dentro do ecossistema **SpotifUM**.

3.2. Carregamento de estados

Para simular um cenário realista de utilização contínua da aplicação **SpotifUM**. O sistema foi concebido para recuperar e restaurar automaticamente todos os dados relevantes — como utilizadores, músicas, playlists, e estatísticas de reprodução — refletindo um histórico completo e coerente de interações.

Com esta abordagem, ao carregar o estado da aplicação, é possível observar de imediato informações como playlists geradas automaticamente com base em hábitos de reprodução passados, músicas previamente reproduzidas por diferentes utilizadores, e estatisticas de conteúdos mais populares, sem necessidade de simulação adicional de eventos ou avanço no tempo.

Para tornar este processo mais flexível e modular, foram criadas funcoes para salvar e carregar esses dados tais como estes usados para as musicas:

```
public void save(String filePath) throws IOException {
      try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filePath))) {
          out.writeObject(getAllMusics());
      }
  }

  @SuppressWarnings("unchecked")
  public void load(String filePath) throws IOException, ClassNotFoundException {

      List<Musica> musicas;
      try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filePath))) {
          musicas = (List<Musica>) in.readObject();
      }
      MusicManager manager = new MusicManager();
      for (Musica m : musicas) {
          manager.insertMusica(m);
      }
      this.musicas.clear();
      this.musicas.putAll(manager.musicas);
  }
```

3.3. Realização de estatísticas sobre o estado do programa

O ***QueriesManager*** é a classe responsável na implementação da resolução de estatísticas da aplicação. Esta classe acessa a informação dos utilizadores, das musicas e das playlists . Assim, sempre que ocorrer uma alteração temporal ou forem adicionadas novas informações, os resultados serão recalculados.

Todas as estatísticas pedidas foram implementadas:
1. Qual é a música mais reproduzida?
2. Qual é o intérprete mais escutado?
3. Qual o utilizador que mais músicas ouviu num período ou desde sempre?
4. Qual é o utilizador com mais pontos?
5. Qual o tipo de música mais reproduzida?
6. Quantas playlists públicas é que existem?
7. Qual o utilizador que tem mais playlists.

A resolução de todas as estatísticas segue um procedimento uniforme. Ao recebermos a entrada de dados, o primeiro passo é verificar a sua validade. Após validar os dados, procedemos a uma análise abrangente, percorrendo todas as informações relevantes. É importante destacar que este processo é iterativo. Por fim, ao atualizarmos ou adicionarmos novos dados, e logo de seguida os métodos são chamados e as estatísticas são calculadas.

3.4. Criação da Música Explícita

Com o objetivo de representar músicas com conteúdo sensível ou impróprio para determinados públicos, surgiu a necessidade de introduzir a noção de **Música Explícita** no sistema. Tal como feito anteriormente com outras entidades especializadas, procurámos manter uma abordagem modular e extensível.

Para isso, criámos a classe `Explicita`, que herda diretamente da classe base `Musica`. Esta especialização permite identificar, criar e manipular músicas do tipo explícito de forma isolada, sem impactar o comportamento das restantes músicas no sistema.

A classe `Explicita` reutiliza toda a lógica da classe `Musica`, herdando os seus atributos e métodos, mas distingue-se semanticamente por representar uma categoria específica. Esta diferenciação pode ser aproveitada no futuro para aplicar políticas distintas, como restrições de idade ou filtragens automáticas em playlists e recomendações.

Com esta estratégia, garantimos que a introdução de novas funcionalidades associadas a músicas explícitas possa ser feita sem alterar o comportamento das restantes classes, respeitando assim os princípios de encapsulamento e modularidade.

3.5. Geração de uma Playlist com Critérios Definidos

Como requisito adicional, o sistema permite gerar automaticamente uma playlist personalizada para um utilizador com base em critérios específicos. Entre os critérios possíveis estão:

1. a inclusão de músicas dentro dos géneros preferidos do utilizador,
2. a limitação da duração total da playlist,
3. e a filtragem apenas por músicas do tipo `Explicita`.

Estes critérios visam adaptar a experiência musical às preferências e necessidades do utilizador, respeitando ainda restrições impostas, como o tipo de subscrição do utilizador (exclusivo para utilizadores premium).

Durante o desenvolvimento desta funcionalidade, surgiu a necessidade de filtrar e selecionar músicas com base nos critérios definidos, garantindo que a playlist gerada seja coerente e relevante. Para facilitar a gestão e evolução desta lógica, optámos por criar uma classe específica — `GeneratedMusics` — que encapsula os diferentes métodos de geração da playlist, permitindo modularidade, clareza e facilidade de manutenção caso surjam novas exigências futuras.

3.6. Salvaguarda do estado da aplicação

Para garantir a continuidade e conveniência para o utilizador, implementámos uma funcionalidade de salvaguarda do estado da aplicação, utilizando técnicas de serialização das instâncias e escrita em binário nos ficheiros. Isto permite que o utilizador retome a sua atividade, mesmo após fechar e reabrir a aplicação. Todos os dados relacionados com os musicas, playlists ,sistema de pontos. Com esta funcionalidade, proporcionamos uma experiência contínua e sem preocupações para todos os nossos utilizadores.

4. Utilização da aplicação

Nesta secção, abordamos a forma de utilização da nossa aplicação, ajudando no processo desde como iniciar, até às funcionalidades presentes e como elas operam.

4.1. Funcionamento geral

A aplicação segue uma abordagem sequencial através de menus de linha de comando (CLI), que nos permite navegar pelas restantes funcionalidades da aplicação. Esta estrutura de menus simplifica a interação do utilizador com o sistema, fornecendo uma experiência intuitiva e direcionada. Ao utilizar os menus, os utilizadores podem facilmente explorar e aceder às diversas funcionalidades disponíveis na aplicação. Além disso, a natureza sequencial dos menus promove uma navegação fluida e organizada, facilitando a utilização e compreensão do sistema

4.2. Início da aplicação

Para iniciar a aplicação, é essencial verificar se todas as dependências estão instaladas, incluindo o Gradle, JUnit5 e JavaDoc. Posteriormente, procede-se à compilação da aplicação utilizando o comando específico, conforme indicado no ficheiro README.md que acompanha o projeto. Uma vez compilada com sucesso, a aplicação está pronta para ser executada através do segundo comando fornecido no mesmo ficheiro. Esta abordagem garante que a aplicação seja executada de forma correta e sem problemas relacionados com dependências ausentes ou compilação inadequada. Comandos para compilar e executar a aplicação, respetivamente:

`gradle build` `java -jar build/libs/spotifum-1.0-SNAPSHOT.jar`

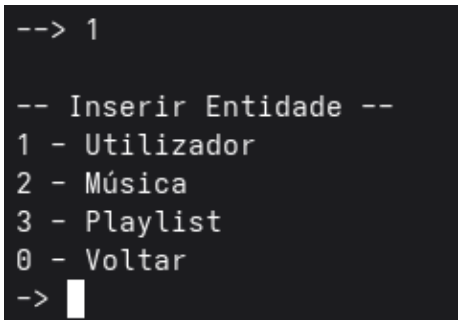
Ao iniciar o programa, é-nos apresentado o seguinte menu principal, com várias funcionalidades disponíveis. Através da primeira opção, podemos inserir novas entidades no sistema. A segunda permite consultar entidades previamente registadas. A terceira serve para apagar entidades existentes. A quarta opção possibilita o registo de reproduções.

Além disso, a aplicação permite guardar o estado atual para um ficheiro (opção 5) e carregar esse estado posteriormente (opção 6). A sétima opção apresenta estatísticas relevantes sobre os dados registados. Já a oitava permite gerar automaticamente uma playlist com base nos critérios definidos. Por fim, a nona opção encerra a aplicação.

```
==== Spotifum G-26 - Menu Principal ====
1 - Inserir Entidade
2 - Consultar Entidade
3 - Apagar Entidade
4 - Registar Reprodução
5 - Guardar Estado para Ficheiro
6 - Carregar Estado de Ficheiro
7 - Estatísticas
8 - Gerar Playlist Automática
0 - Sair
====================================
-->
```

4.3. Inserir Entidade

A imagem apresentada mostra um menu de criação de entidades no sistema Spotifum. Este menu permite ao utilizador inserir novas entidades no sistema, escolhendo entre três opções principais:



Ao selecionar uma das opções, o utilizador é conduzido para um submenu onde deve inserir os dados correspondentes à entidade escolhida.

4.3.1. Utilizador

A criação de um novo utilizador requer a introdução dos seguintes campos:

- **Nome**: nome completo do utilizador.
- **Email**: endereço de email, usado como identificador único.
- **Morada**: localização do utilizador.
- **Plano de Subscrição**: tipo de subscrição associada ao utilizador.

4.3.2. Música

Para inserir uma nova música no sistema, o utilizador deve fornecer:

- **Nome**: título da música.
- **Intérprete**: artista ou banda responsável pela música.
- **Editora**: empresa responsável pela publicação.
- **Letra**: conteúdo textual da música.
- **Partitura** (opcional): ficheiro ou referência à partitura da música.
- **Género**: um dos seguintes valores:
  - Pop
  - Rock
  - Clássica
  - Jazz
  - HipHop
  - Eletrónica
  - Reggae
  - Fado
  - Outro
- **Duração**: duração total da música.
- **Tipo de Música**: pode ser uma das seguintes categorias:
  - Explícita
  - Multimedia
  - Normal

4.3.3. Playlist

A criação de uma nova playlist exige os seguintes dados:

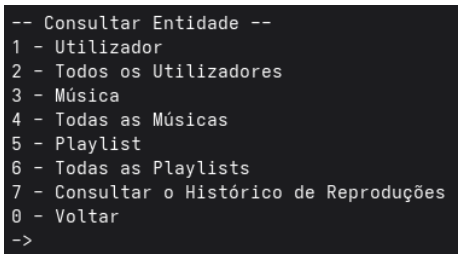
- **Email do utilizador**: email do utilizador proprietário da playlist.

- **Nome**: nome da playlist.
- **Músicas** (opcional): o utilizador pode adicionar músicas à playlist, identificando-as pelo nome.
- **Pública**: indicação se a playlist é pública ou privada. A resposta deve ser “S” para Sim ou “N” para Não.

Este menu constitui uma funcionalidade central na aplicação, permitindo ao utilizador alimentar o sistema com os dados necessários para a gestão de utilizadores, músicas e playlists.

4.4. Consultar Entidades

A imagem apresentada mostra um menu de **consulta de entidades** no sistema **Spotifum**. Este menu permite ao utilizador aceder a informações previamente registadas na aplicação, oferecendo várias opções de visualização e consulta detalhada:



Ao selecionar uma das opções, o utilizador é conduzido para uma interface de consulta específica, onde pode visualizar os dados associados à entidade escolhida.

4.4.1. Utilizador

Permite consultar os dados de um utilizador individual. Será solicitado o email para aceder às suas informações, como nome, email , morada, pontos , plano de subscrição , musicas favoritas e reproduções.

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 1
Email: e1
Utilizador encontrado:
{ nome: "António", email: "e1", morada: "Algures", pontos: 100.00, plano: "PremiumTop",
musicasFavoritas: [], reproducoes: [] }
```

4.4.2. Todos os Utilizadores

Apresenta uma lista completa de todos os utilizadores registados no sistema. Esta opção é útil para visualizar o estado global da base de utilizadores apresentando todas as informações dos utilizadores .

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 2
Todos os Utilizadores:
{ nome: "António", email: "e1", morada: "Algures", pontos: 100.00, plano: "PremiumTop",
musicasFavoritas: [], reproducoes: [] }
{ nome: "João", email: "e2", morada: "Nenhures", pontos: 0.00, plano: "FREE",
musicasFavoritas: [], reproducoes: [] }
```

4.4.3. Música

Oferece a possibilidade de consultar os detalhes de uma música específica. É solicitado o nome da musica e é apresentado o seu nome, intérprete, editora, género, duração, reproduções , letra e partitura.

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 3
Nome: m1
Música encontrada:
{ nome: "m1", interprete: "Toy", editora: "Golden Records", genero: "POP", duracao:
121, reproducoes: 0, letra: "Solta-te, liberta-Te Abre as asas do sonho Tens todo o
futuro para saborear Solta- te, liberta- te Não há nada melhor Depois de um pesadelo
poder acordar Vou beijar, vou dançar Vou hum hum até me cansar Toda a noite, toda a
noite", partitura: [do, re, mi] }
```

4.4.4. Todas as Músicas

Mostra uma listagem completa de todas as músicas disponíveis no sistema, apresenta as informações de todas as musicas disponiveis.

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 4
Todas as Músicas:
{ nome: "m1", interprete: "Toy", editora: "Golden Records", genero: "POP", duracao:
121, reproducoes: 0, letra: "Solta-te, liberta-Te Abre as asas do sonho Tens todo o
futuro para saborear Solta- te, liberta- te Não há nada melhor Depois de um pesadelo
poder acordar Vou beijar, vou dançar Vou hum hum até me cansar Toda a noite, toda a
noite", partitura: [do, re, mi] }
{ nome: "m2", interprete: "Rosinha", editora: "Local Tunes", genero: "CLASSICA",
duracao: 140, reproducoes: 0, letra: "Meu amor trabalha muito Precisa de comer Eu faço
sempre um lanche P'ra ele não enfraquecer Preparo um pacote Ponho de tudo um bocadinho
E na hora de levar Eu ponho-me a caminho Eu levo no pacote Ai eu levo sim senhor Eu
levo no pacote Ai tem outro sabor", partitura: [mi, re, do] }
{ nome: "m3", interprete: "Toni Carreira", editora: "Hits Producer", genero: "OUTRO",
duracao: 100, reproducoes: 0, letra: "Lembro-me de uma aldeia perdida na beira A terra
que me viu nascer Lembro-me de um menino que andava sozinho Sonhava vir um dia a ser
Sonhava ser cantor de cantigas de amor Com a força de Deus venceu Dessa pequena aldeia
o menino era eu", partitura: [la, la, mi] } Vídeo: https://youtu.be/Sx_OALoIrj8
```

4.4.5. Playlist

Permite visualizar os detalhes de uma playlist específica sendo que recebe o emial do Utilizador dono da playlist e o seu nome e dá display do seu nome, criador , se é pública ou privada ,reproduções e as musicas pertencenetes a essa playlist .

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 5
Email do Utilizador: e1
Nome da Playlist: p1
Playlist encontrada:
{ nome: "p1", criador: "e1", publica: true, reproducoes: 0, musicas: ["m1", "m2",
"m3"] }
```

4.4.6. Todas as Playlists

Apresenta todas as playlists existentes no sistema, independentemente do utilizador que as criou

```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 6
Todas as Playlists:
{ nome: "p1", criador: "e1", publica: true, reproducoes: 0, musicas: ["m1", "m2",
"m3"] }
{ nome: "favorite_musics", criador: "e1", publica: false, reproducoes: 0, musicas: [] }
{ nome: "p2", criador: "e2", publica: false, reproducoes: 0, musicas: ["m1"] }
```

4.4.7. Consultar o Histórico de Reproduções

Permite aceder ao histórico de músicas reproduzidas por um determinado utilizador. Esta funcionalidade é útil para analisar padrões de consumo musical.

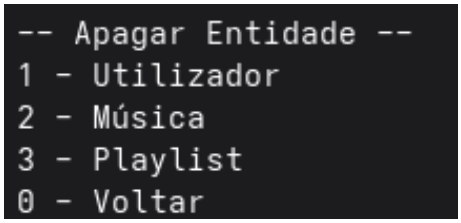
```
-- Consultar Entidade --
1 - Utilizador
2 - Todos os Utilizadores
3 - Música
4 - Todas as Músicas
5 - Playlist
6 - Todas as Playlists
7 - Consultar o Histórico de Reproduções
0 - Voltar
-> 7
Email: e1
Todas as Reproduções do Utilizador:
{ musica: "m1", data: "2025-05-17T17:39:02.893264958", email: "e1" }
{ musica: "m1", data: "2025-05-17T17:39:30.001547485", email: "e1" }
{ musica: "m2", data: "2025-05-17T17:39:43.993061682", email: "e1" }
{ musica: "m2", data: "2025-05-17T17:39:54.414690866", email: "e1" }
```

4.4.8. Voltar

Retorna ao menu anterior, permitindo ao utilizador sair do modo de consulta e regressar ao menu principal.

4.5. Apagar Entidade

O menu apresentado na imagem permite ao admin eliminar entidades previamente registadas no sistema **Spotifum**. Esta operação é irreversível e deve ser usada com precaução, pois os dados eliminados não podem ser recuperados posteriormente.



4.5.1. Utilizador

Se o admin optar por apagar um utilizador, será solicitado o **email** correspondente. Por exemplo:

```
-- Apagar Entidade --
1 - Utilizador
2 - Música
3 - Playlist
0 - Voltar
-> 1
Email: e1
Utilizador apagado com sucesso!
```

4.5.2. Música

Ao escolher apagar uma música, será pedido o **nome da música** a remover.Por exemplo:

```
-- Apagar Entidade --
1 - Utilizador
2 - Música
3 - Playlist
0 - Voltar
-> 2
Nome: m1
Música apagada com sucesso!
```

4.5.3. Playlist

Para apagar uma playlist, o sistema requer dois dados:

- O **email do utilizador** que criou a playlist.
- O **nome da playlist**.

```
-- Apagar Entidade --
1 - Utilizador
2 - Música
3 - Playlist
0 - Voltar
-> 3
Email do Utilizador: e1
Nome da Playlist: p1
Playlist apagada com sucesso!
```

4.6. Menu de Registo de Reprodução

O menu **Registar Reprodução** do sistema **Spotifum** permite ao utilizador simular a reprodução de músicas, playlists ou músicas favoritas, associando estas ações ao seu perfil. Esta funcionalidade é essencial para acompanhar os hábitos de escuta e fornecer estatísticas relevantes.

O menu principal apresenta a seguinte opção:

```
→ 4 - Registar Reprodução
```

Ao selecionar esta opção, o utilizador tem acesso ao seguinte submenu:

4.7. Reproduzir

Este submenu permite escolher o tipo de conteúdo a reproduzir:

```
1. Uma Música
2. Playlist
3. Músicas Favoritas
4. Playlist Gerada Automaticamente
0. Voltar
```

4.7.1. Uma Música

O sistema solicita dois dados:

- O **email do utilizador**.
- O **nome da música** a reproduzir.

Exemplo:

```
▶ A reproduzir: `m1`
🪄 Toy | 🎼 POP | ⏱ 121s

Durante a reprodução, a letra da música é apresentada linha a linha. No final, é feita
uma pergunta:

> Deseja adicionar esta música aos favoritos?
> S/N: N
```

Esta interação permite ao utilizador guardar a música nos seus favoritos, caso deseje.

4.7.2. Playlist

O utilizador insere:

- O **email do utilizador**.
- O **nome da playlist** a reproduzir.

São então reproduzidas todas as músicas da playlist, uma a uma. Após cada música, o sistema apresenta os seguintes controlos:

```
- 1 - Próxima música
- 2 - Música anterior
- 3 - Repetir música atual
- 4 - Adicionar ou remover dos favoritos
- 0 - Parar reprodução
```

Estes controlos permitem uma navegação interativa durante a reprodução, semelhante a um leitor multimédia.

4.7.3. Músicas Favoritas

Após indicar o **email**, o sistema começa a reproduzir as músicas favoritas do utilizador. Os controlos são os mesmos da reprodução de playlists. E se o utilizador tentar remover uma música que está nos favoritos, será exibida uma mensagem:

```
Música removida dos favoritos com sucesso!
```

4.7.4. Playlist Gerada Automaticamente

O sistema pede o **email do utilizador** e tenta aceder à playlist automática gerada para o mesmo e reproduz tal como uma playlist normal .

4.7.5. Voltar

Esta opção retorna ao menu principal, sem iniciar qualquer reprodução.

4.8. Guardar estado

Após a utilização da aplicação, é sempre possível guardar o estado atual, incluindo todas as informações e registos realizados durante a sessão. Essa informação é armazenada de forma a permitir que, numa próxima utilização da aplicação, o utilizador possa retomar exatamente de onde parou, permitindo-lhe manter o seu progresso e registos ao longo do tempo, mesmo em diferentes sessões de utilização da aplicação.

Opção em questão:	Demonstração de como guardar o estado através do modo utilizador:

`5 - Guardar Estado para Ficheiro`	`--> 5`
	`Insira o local onde guardar o estado (ex: data/Estado1/): Estado2/`
	`Estado guardado com sucesso!`

4.9. Carregar estado

Antes da utilização da aplicação, é sempre possível carregar o estado anterior, incluindo todas as informações e registos realizados durante a sessão. Essa informação é armazenada de forma a permitir que, numa próxima utilização da aplicação, o utilizador possa retomar exatamente de onde parou, permitindo-lhe manter o seu progresso e registos ao longo do tempo, mesmo em diferentes sessões de utilização da aplicação.

Opção em questão:	Demonstração de como guardar o estado através do modo utilizador:

`6 - Carregar Estado de Ficheiro`	`--> 6`
	`Insira o local de onde carregar o estado (ex: data/Estado1/): Estado1/`
	`Ao continuar vai apagar todas as alterações locais feitas até ao momento! Deseja continuar? (S - Continuar / N - Voltar)`
	`S/N: S`
	`Estado carregado com sucesso!`

4.10. Terminar aplicação

Assim como em qualquer aplicação convencional, também é disponibilizada uma opção para sair do programa. Ao selecionar esta opção, o utilizador é encaminhado de volta ao menu inicial da aplicação. referido na Inicio_da_aplicação

Opção em questão:	Demonstração de como guardar o estado através do modo utilizador:
`0 - Sair`	`--> 0` `A sair... Obrigado por usar o Spotifum G-26!`

4.11. Execução de estatísticas

O administrador tem a possibilidade de realizar estatísticas

`7 - Estatísticas`

Assim sendo mesmo pode obter um maior conhecimento sobre a sua aplicação. A análise estatística permite ao administrador extrair informações valiosas, sobre o desempenho, utilização e tendências da

aplicação. Essa compreensão mais profunda da aplicação pode ajudar o administrador a tomar decisões informadas e aprimorar continuamente a experiência do utilizador, bem como otimizar o funcionamento geral da aplicação. Sendo assim no inicio tem uma questão sendo que corresponde á estatistica :

+ Utilizador que mais músicas ouviu

Onde está tanto pode ser em todo o periodo de tempo como entre dois intervalos.

Opção em questão:

```
Deseja especificar um intervalo de tempo
para o utilizador que mais músicas
ouviu? (S/N)
S/N: N
```

Demonstração de das estatísticas:

```
[Música Mais Reproduzida] - m1 - Toy - 7
reproduções
[Intérprete Mais Escutado] - Toy
[Utilizador que mais músicas ouviu] - e1
[Utilizador com mais Pontos] - e1
[Tipo de Música Mais Reproduzida] - POP
[Número de Playlists Públicas] - 1
[Utilizador com mais Playlists] - e1
```

5. Conclusão

A aplicação Spotifum foi desenvolvida como uma plataforma robusta para a gestão de conteúdos musicais. Suporta a criação e gestão de entidades fundamentais como utilizadores, músicas e playlists, permitindo uma experiência personalizada e dinâmica.

Para além da reprodução de conteúdos musicais, o sistema oferece funcionalidades avançadas como a geração automática de playlists com base nas preferências dos utilizadores, podendo incluir restrições como tempo máximo de reprodução ou filtragem por tipo de música (por exemplo, apenas músicas explícitas).

A aplicação também disponibiliza estatísticas relevantes sobre a atividade dos utilizadores e os conteúdos mais reproduzidos, oferecendo uma visão detalhada da interação com a plataforma.

Em suma, o Spotifum combina gestão eficiente com personalização musical, proporcionando uma experiência completa, intuitiva e adaptada às preferências individuais de cada utilizador.

Bibliografia

[1] «Abstrat Factory Design Pattern». Acedido: 4 de abril de 2025. [Online]. Disponível em: https://refactoring.guru/design-patterns/abstract-factory

[2] «Facade Design Pattern». Acedido: 4 de abril de 2025. [Online]. Disponível em: https://refactoring.guru/design-patterns/facade

[3] «Observer Design Pattern». Acedido: 5 de abril de 2025. [Online]. Disponível em: https://refactoring.guru/design-patterns/observer

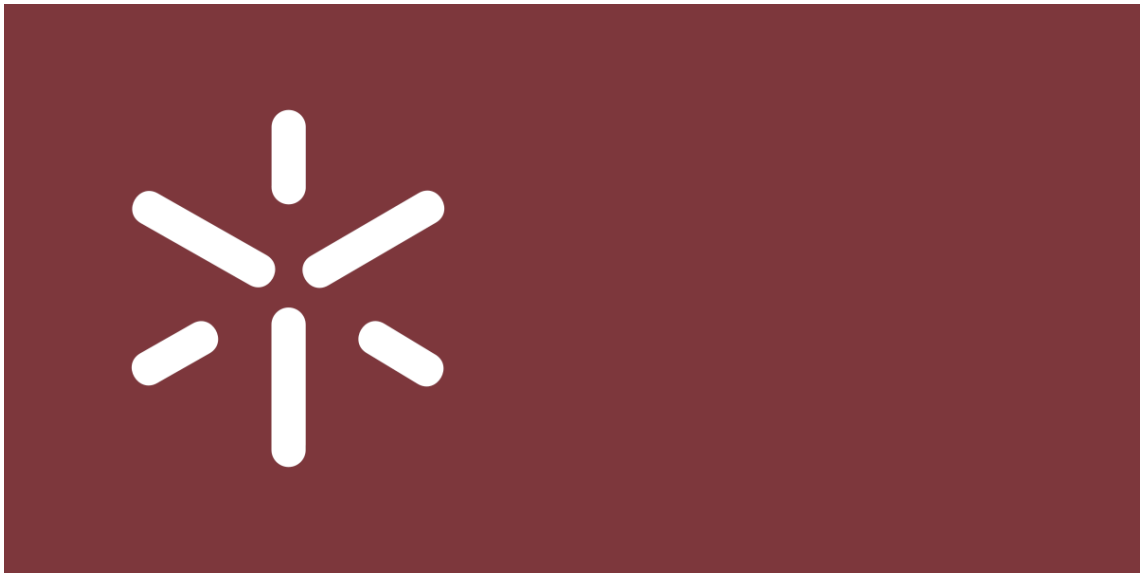
[4] «Visual Paradigm Documentation». Acedido: 16 de abril de 2025. [Online]. Disponível em: https://www.visual-paradigm.com/support/documents/

Lista de Siglas e Acrónimos

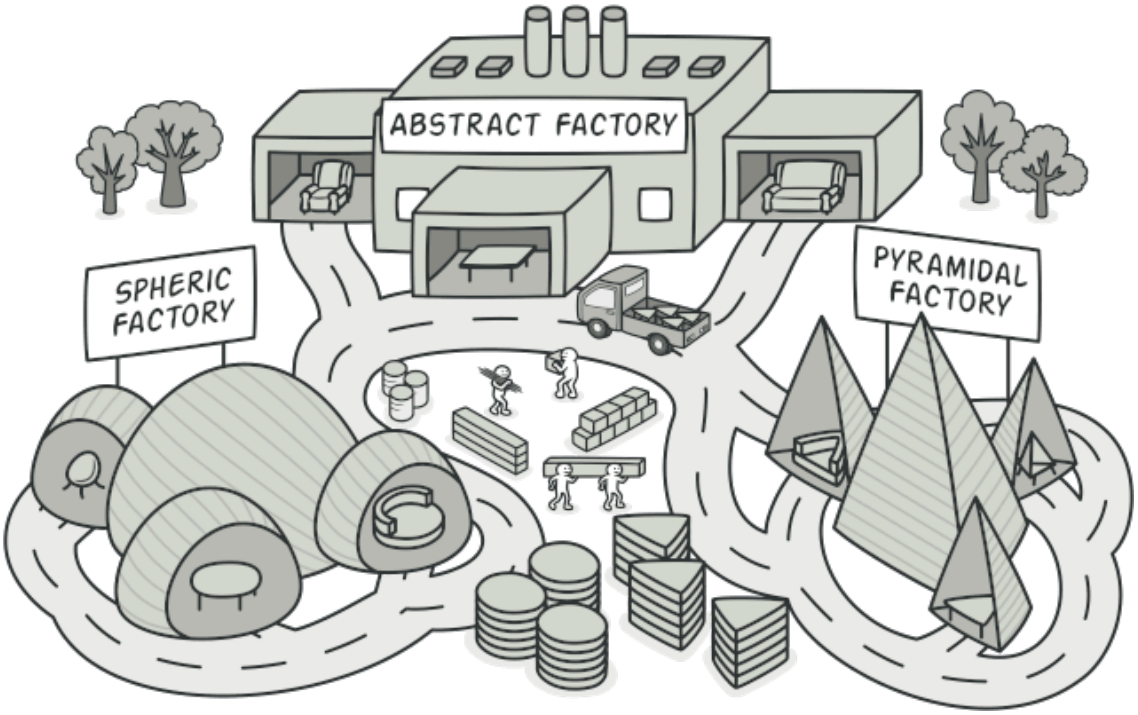
POO Programação Orientada aos Objetos
UML Unified Modeling Language
CLI Command Line Interface

Anexos

Anexo 1: Logo da Universidade do Minho.

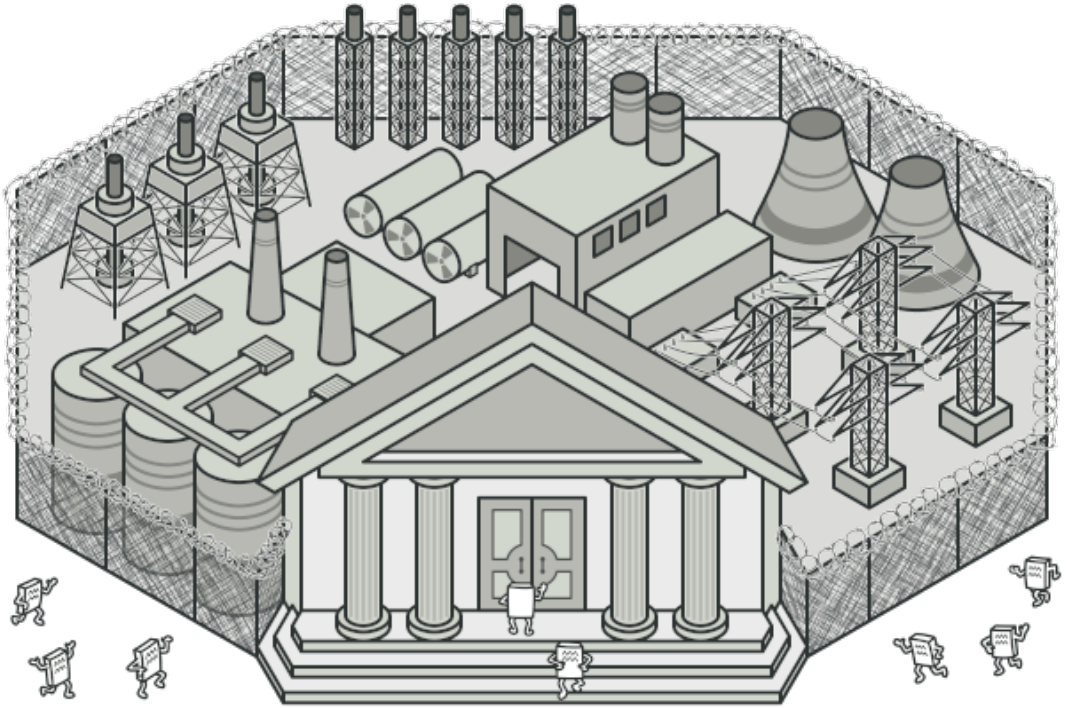


Anexo 2: Exemplo de representação gráfica do padrão Abstract Factory.

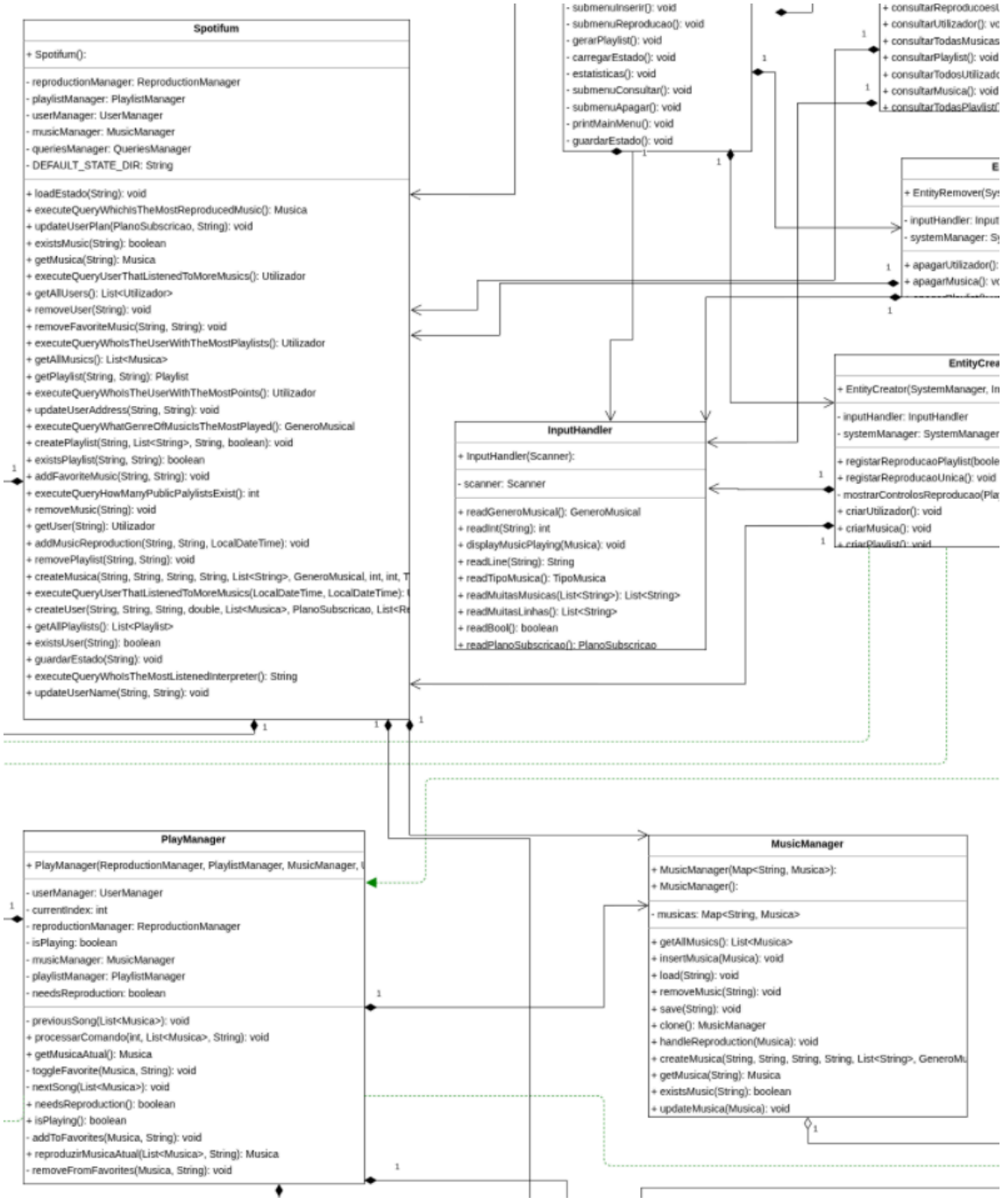


Anexo 3: Exemplo de representação concetual do padrão Abstract Factory na aplicação.

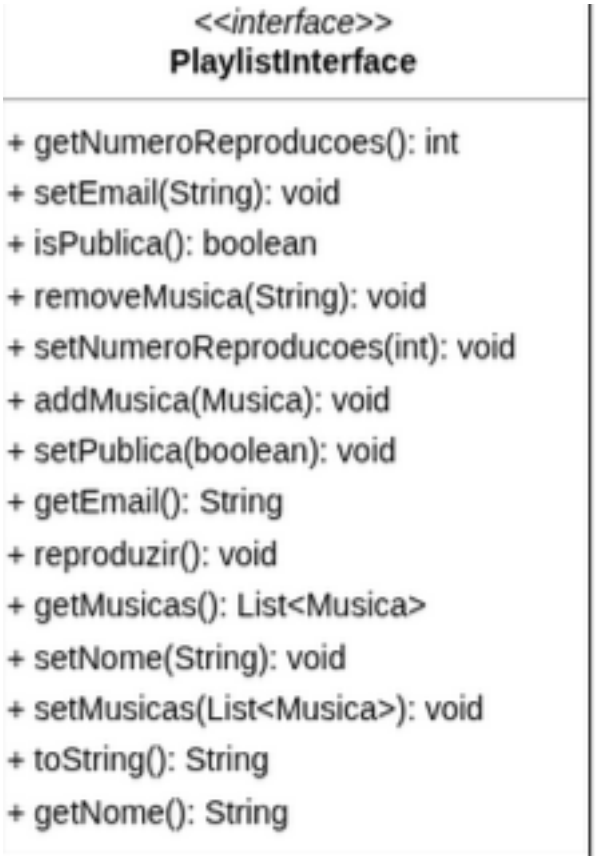
Anexo 4: Exemplo de representação gráfica do padrão Facade.



Anexo 5: Exemplo de representação concetual do padrão Facade na aplicação.

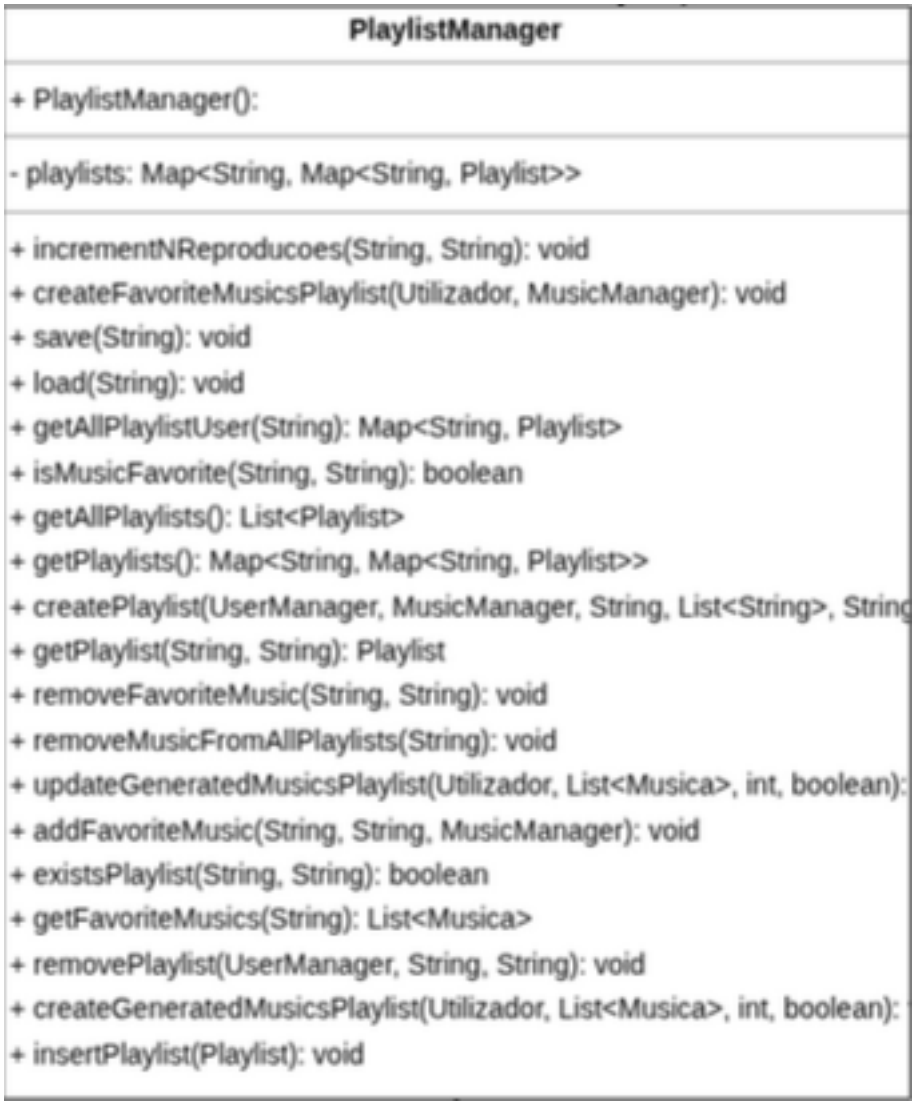


Anexo 6: Representação da interface "PlaylistInterface" no diagrama de classes.

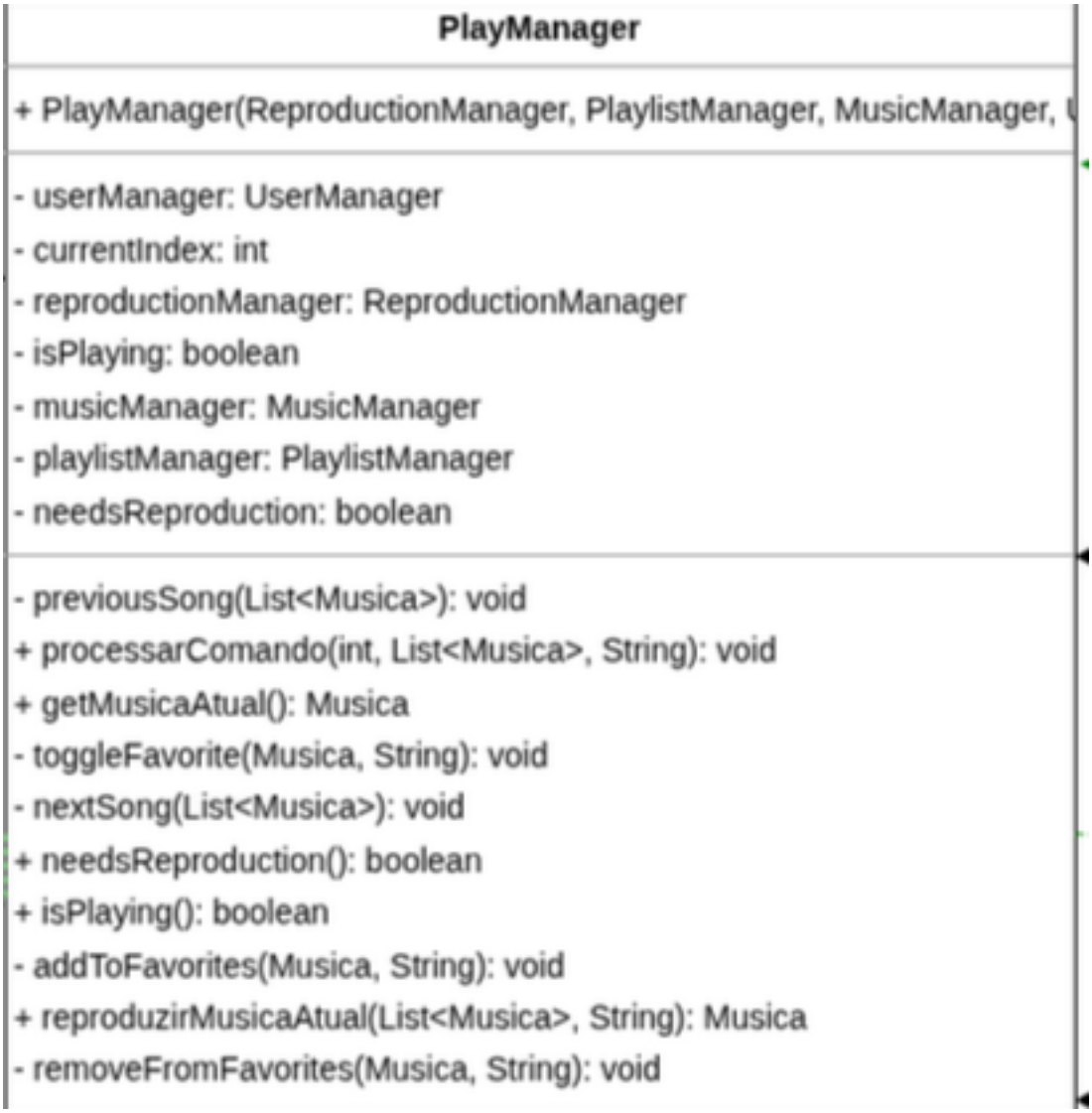


Anexo 7: Representação da classe concreta "Playlist" no diagrama de classes.

Anexo 8: Representação da classe concreta "PlaylistManager" no diagrama de classes.

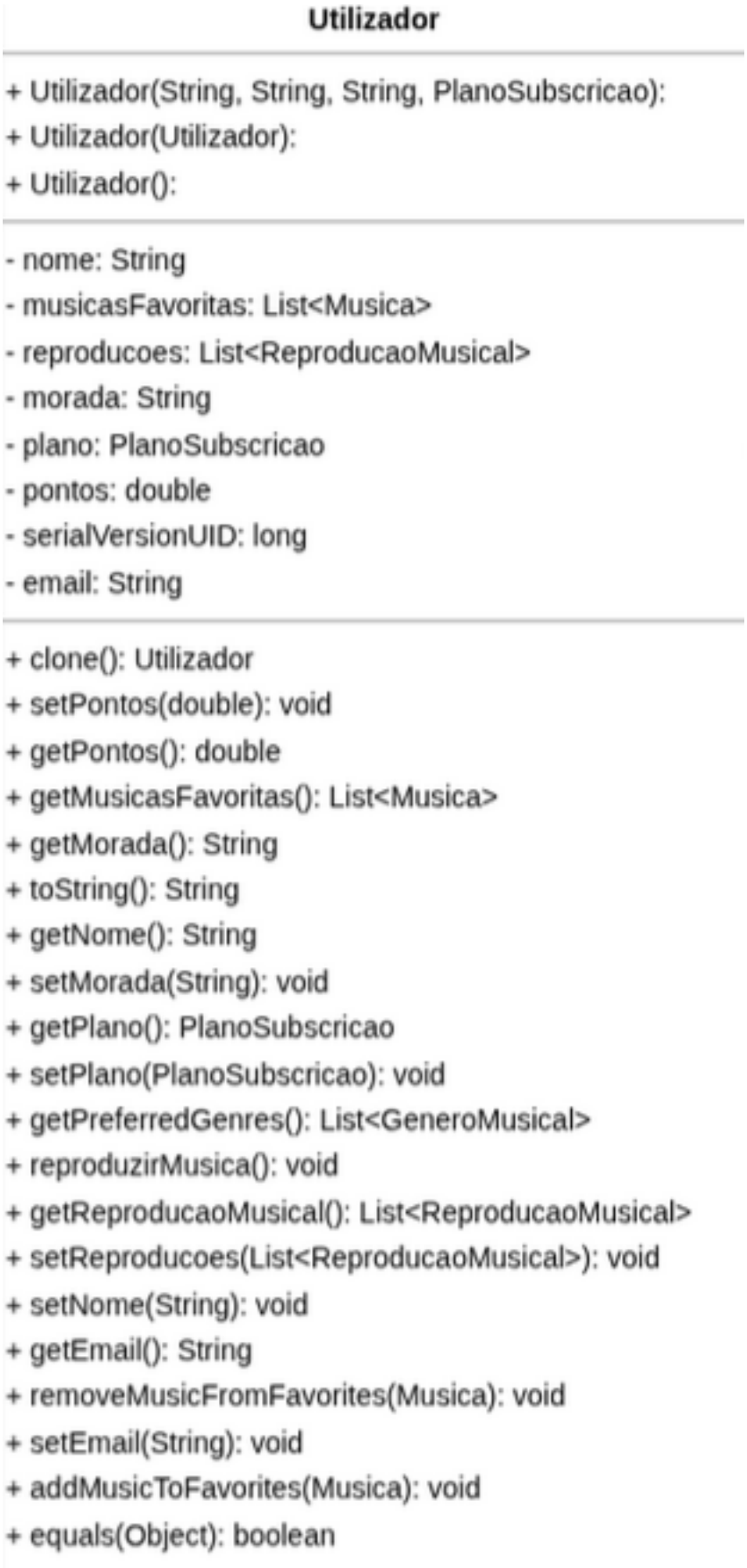


Anexo 9: Representação da classe concreta "PlayManager" no diagrama de classes.

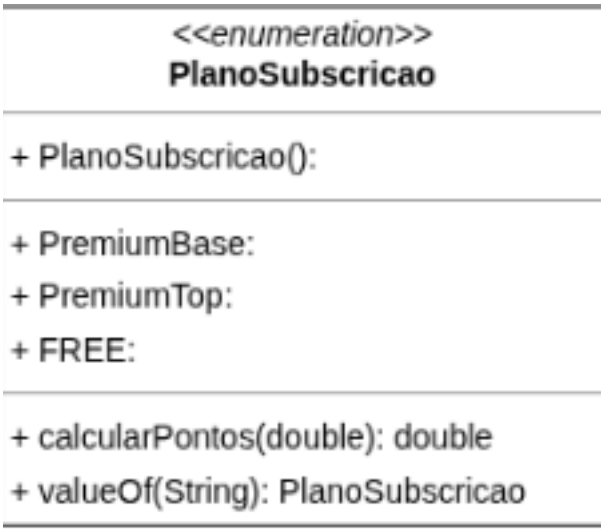


Anexo 10: Representação da interface "UtilizadorInterface" no diagrama de classes.

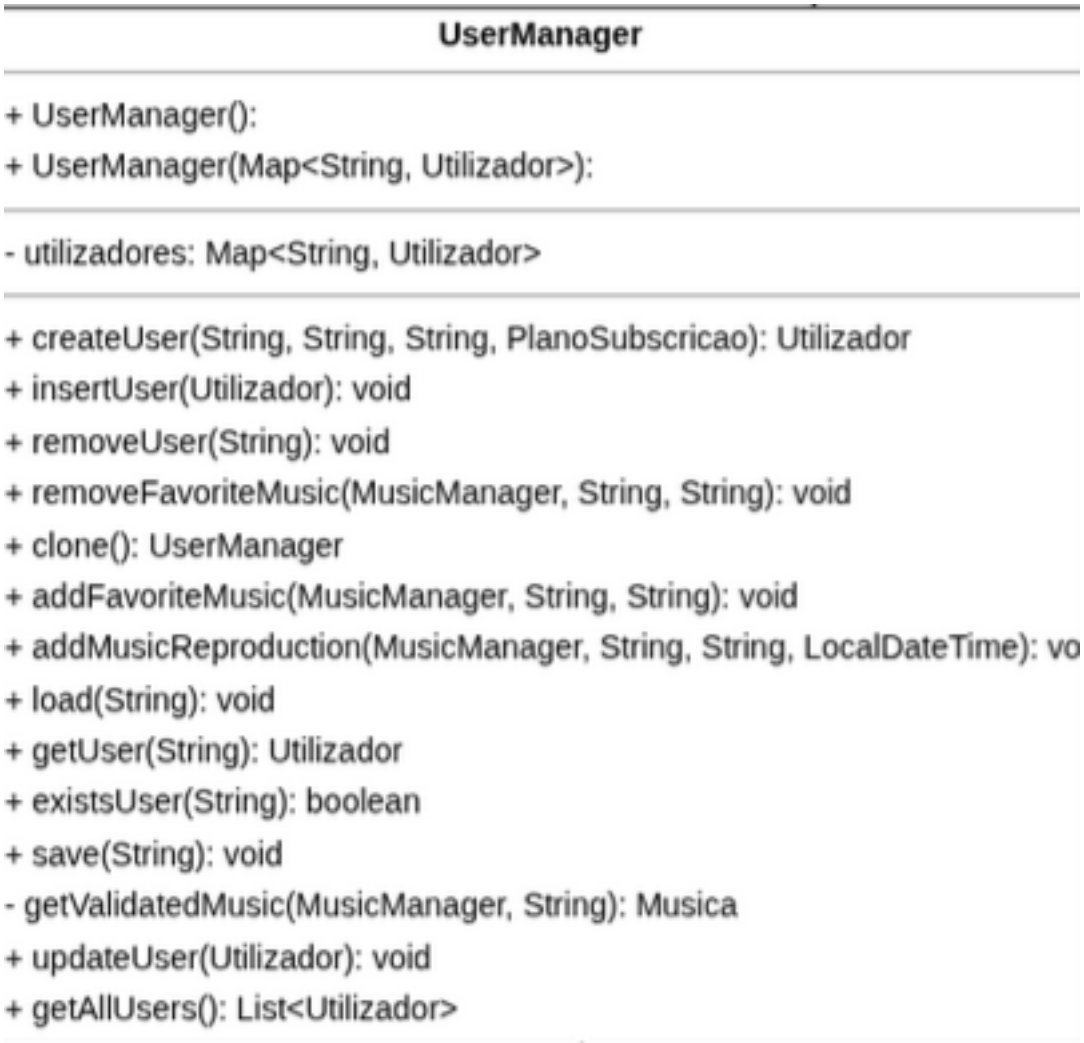
Anexo 11: Representação da classe concreta "Utilizador" no diagrama de classes.


Utilizador
+ Utilizador(String, String, String, PlanoSubscricao):
+ Utilizador(Utilizador):
+ Utilizador():
- nome: String
- musicasFavoritas: List<Musica>
- reproducoes: List<ReproducaoMusical>
- morada: String
- plano: PlanoSubscricao
- pontos: double
- serialVersionUID: long
- email: String
+ clone(): Utilizador
+ setPontos(double): void
+ getPontos(): double
+ getMusicasFavoritas(): List<Musica>
+ getMorada(): String
+ toString(): String
+ getNome(): String
+ setMorada(String): void
+ getPlano(): PlanoSubscricao
+ setPlano(PlanoSubscricao): void
+ getPreferredGenres(): List<GeneroMusical>
+ reproduzirMusica(): void
+ getReproducaoMusical(): List<ReproducaoMusical>
+ setReproducoes(List<ReproducaoMusical>): void
+ setNome(String): void
+ getEmail(): String
+ removeMusicFromFavorites(Musica): void
+ setEmail(String): void
+ addMusicToFavorites(Musica): void
+ equals(Object): boolean

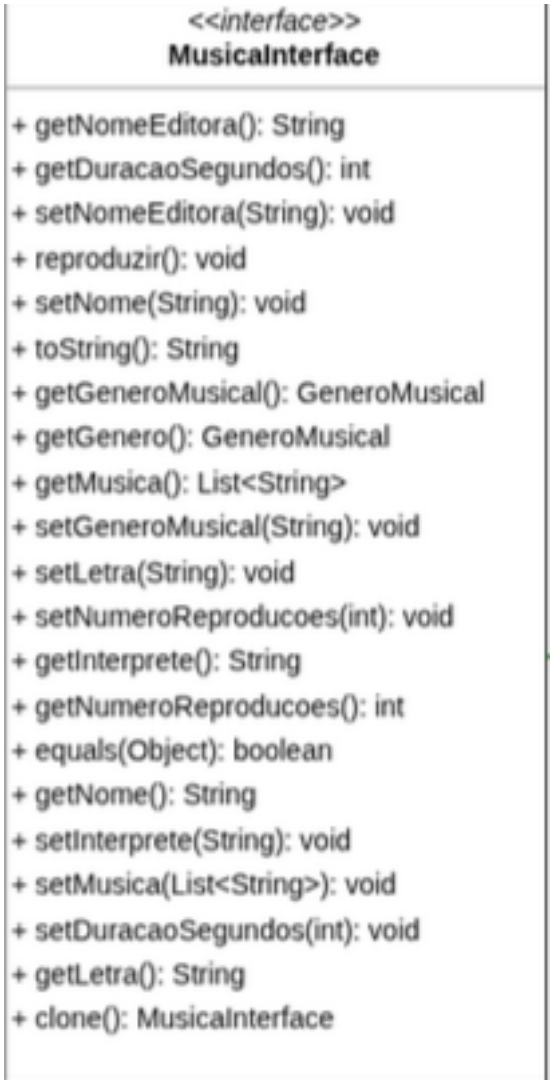
Anexo 12: Representação do enum "PlanoSubscricao" no diagrama de classes.



Anexo 13: Representação da classe concreta "UtilizadorManager" no diagrama de classes.

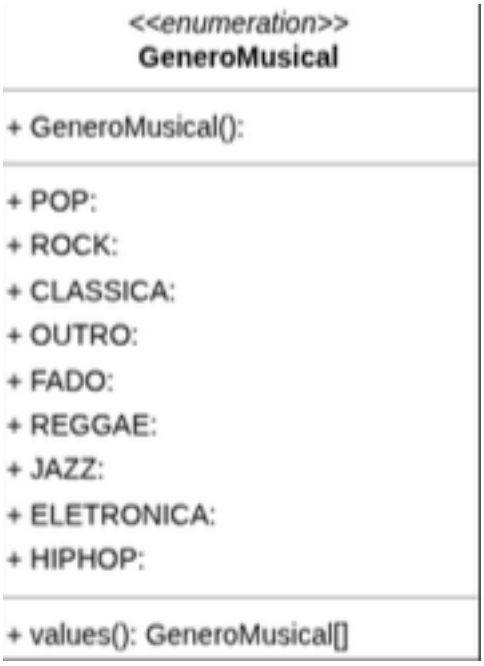


Anexo 14: Representação da interface "MusicaInterface" no diagrama de classes.

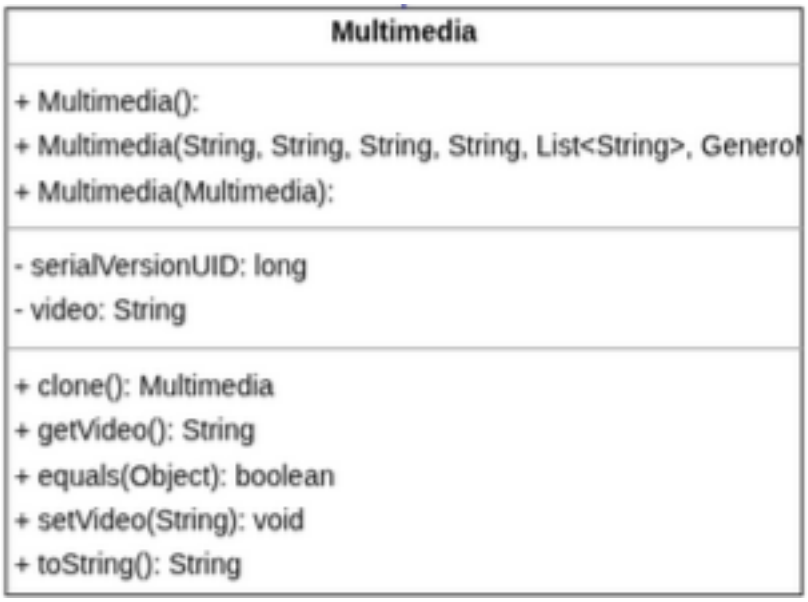


Anexo 15: Representação da classe concreta "Musica" no diagrama de classes.

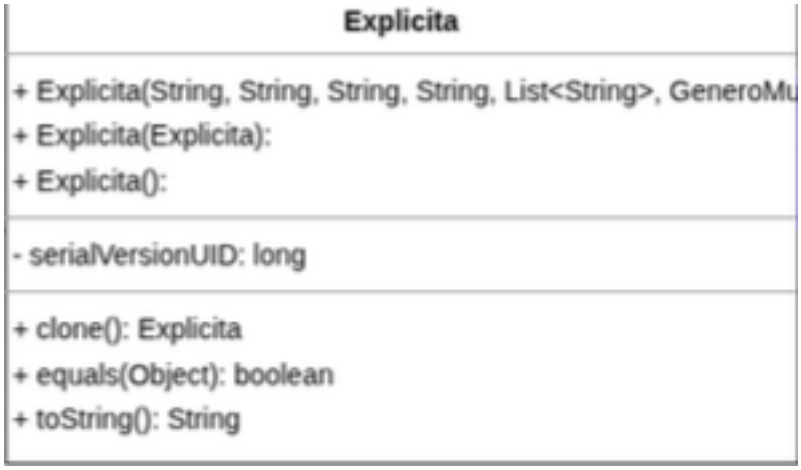
Anexo 16: Representação do enumerável "GeneroMusical" no diagrama de classes.



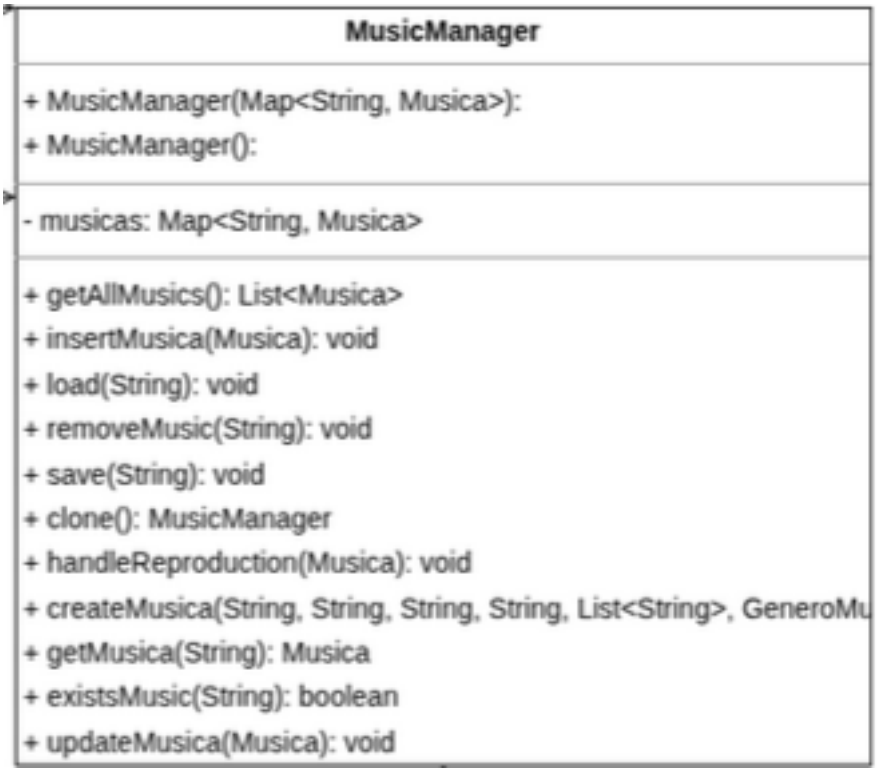
Anexo 17: Representação da subclasse "Multimedia" no diagrama de classes.



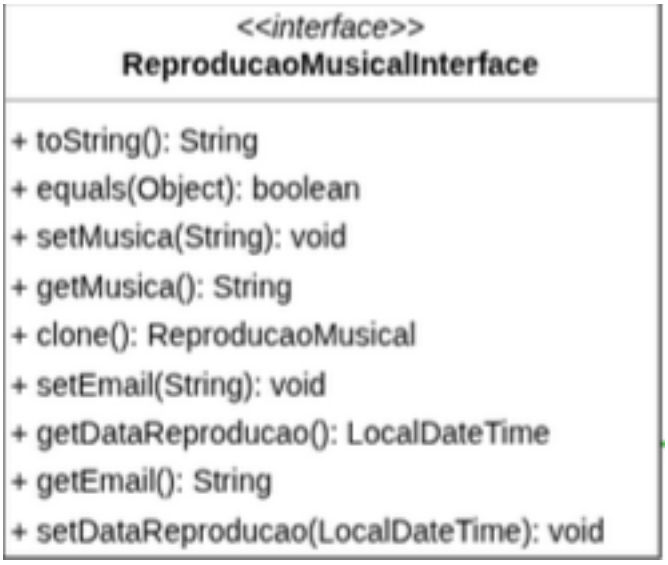
Anexo 18: Representação da subclasse "Explicita" no diagrama de classes.



Anexo 19: Representação da classe concreta "MusicManager" no diagrama de classes.

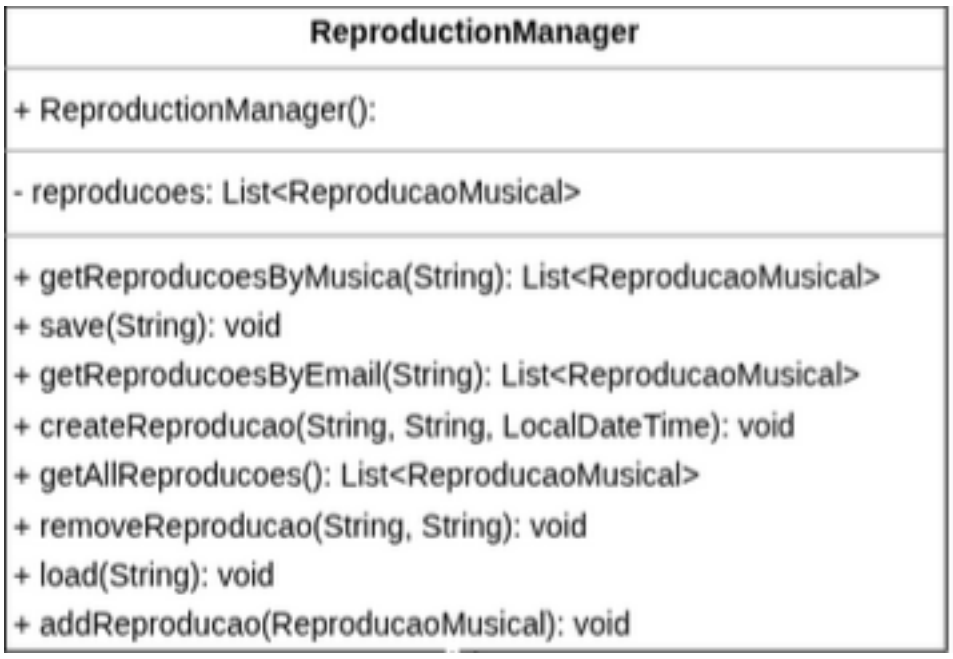


Anexo 20: Representação da interface "ReproducaoMusicalInterface" no diagrama de classes.



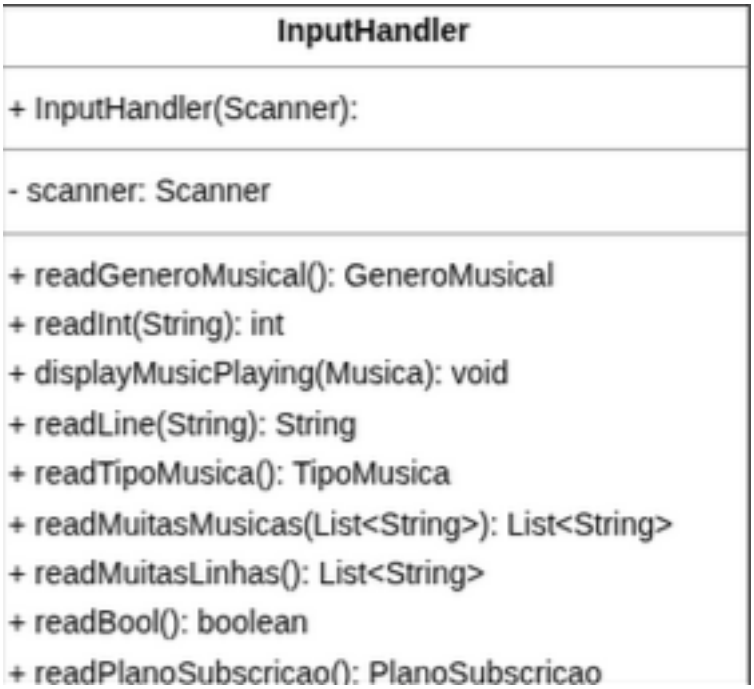
Anexo 21: Representação da classe concreta "ReproducaoMusical" no diagrama de classes.

Anexo 22: Representação da classe concreta "ReproductionManager" no diagrama de classes.

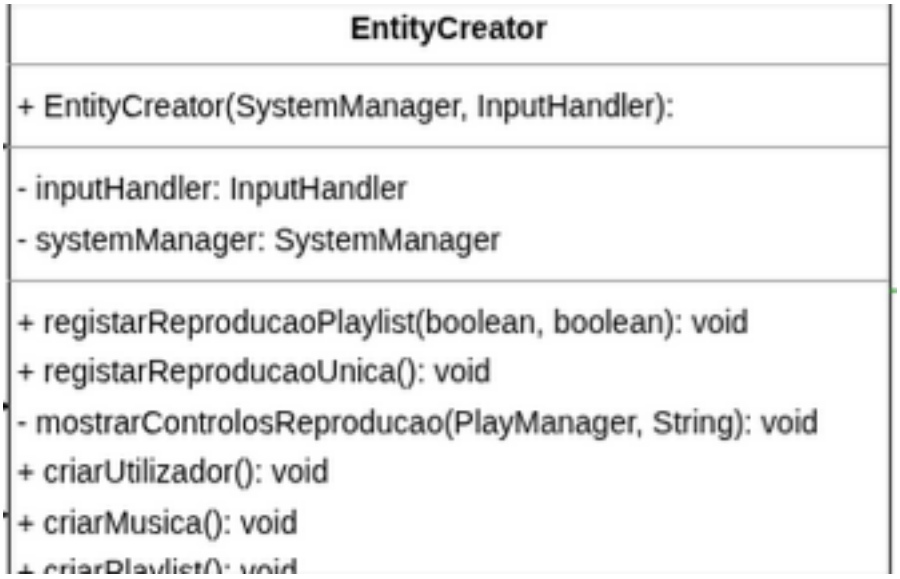


Anexo 23: Representação da classe concreta "Menu" no diagrama de classes.

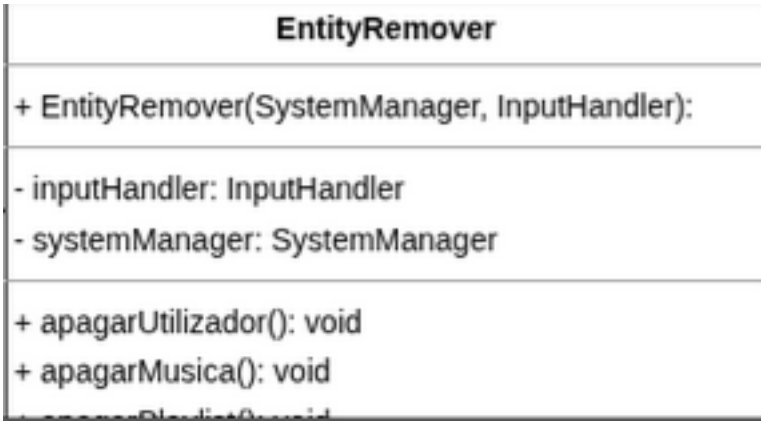
Anexo 24: Representação da classe concreta "InputHandler" no diagrama de classes.



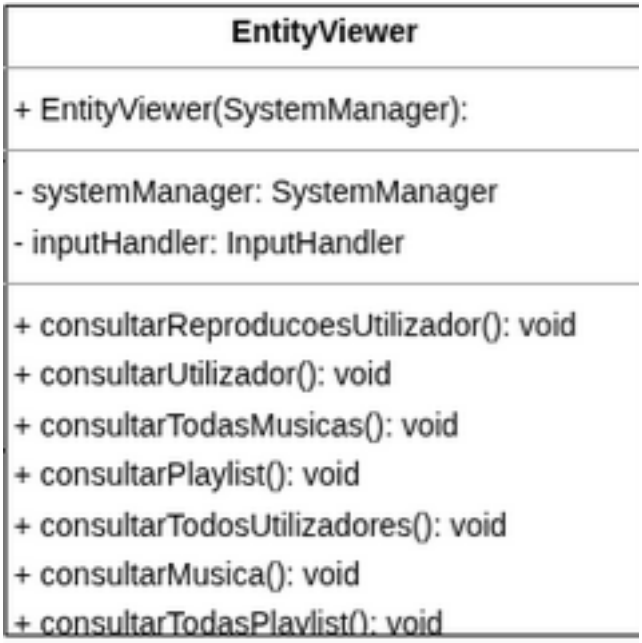
Anexo 25: Representação da classe concreta "EntityCreator" no diagrama de classes.



Anexo 26: Representação da classe concreta "EntityRemover" no diagrama de classes.

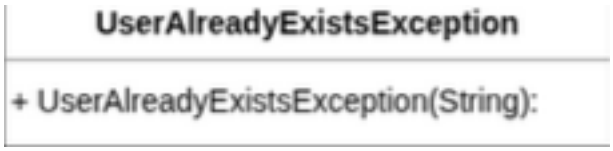


Anexo 27: Representação da classe concreta "EntityViewer" no diagrama de classes.



Anexo 28: Representação da classe concreta "QueriesManager" no diagrama de classes.

Anexo 29: Representação da classe concreta "UserAlreadyExistsException" no diagrama de classes.



Anexo 30: Representação da classe concreta "UserNotFoundException" no diagrama de classes.

Anexo 31: Representação da classe concreta "MusicNotFoundException" no diagrama de classes.

Anexo 32: Representação da classe concreta "Spotifum" no diagrama de classes.


Spotifum
+ Spotifum():
- reproductionManager: ReproductionManager
- playlistManager: PlaylistManager
- userManager: UserManager
- musicManager: MusicManager
- queriesManager: QueriesManager
- DEFAULT_STATE_DIR: String
+ loadEstado(String): void
+ executeQueryWhichIsTheMostReproducedMusic(): Musica
+ updateUserPlan(PlanoSubscricao, String): void
+ existsMusic(String): boolean
+ getMusica(String): Musica
+ executeQueryUserThatListenedToMoreMusics(): Utilizador
+ getAllUsers(): List<Utilizador>
+ removeUser(String): void
+ removeFavoriteMusic(String, String): void
+ executeQueryWhoIsTheUserWithTheMostPlaylists(): Utilizador
+ getAllMusics(): List<Musica>
+ getPlaylist(String, String): Playlist
+ executeQueryWhoIsTheUserWithTheMostPoints(): Utilizador
+ updateUserAddress(String, String): void
+ executeQueryWhatGenreOfMusicIsTheMostPlayed(): GeneroMusical
+ createPlaylist(String, List<String>, String, boolean): void
+ existsPlaylist(String, String): boolean
+ addFavoriteMusic(String, String): void
+ executeQueryHowManyPublicPalylistsExist(): int
+ removeMusic(String): void
+ getUser(String): Utilizador
+ addMusicReproduction(String, String, LocalDateTime): void
+ removePlaylist(String, String): void
+ createMusica(String, String, String, String, List<String>, GeneroMusical, int, int, T
+ executeQueryUserThatListenedToMoreMusics(LocalDateTime, LocalDateTime): U
+ createUser(String, String, String, double, List<Musica>, PlanoSubscricao, List<Re
+ getAllPlaylists(): List<Playlist>
+ existsUser(String): boolean
+ guardarEstado(String): void
+ executeQueryWhoIsTheMostListenedInterpreter(): String
+ updateUserName(String, String): void

Anexo 33: Representação da classe concreta "Main" no diagrama de classes.