Universidade do Minho

Programação Orientada aos Objetos

Trabalho Prático - SpotifUM

Grupo 99 – Ciências da Computação

Ano Letivo 2024/2025

Fábio Alexandre Magalhães Ribeiro

A100058

João Pacheco Mirra

A100083

Índice

Índice de Figuras

1. Introdução

Durante a disciplina de Programação Orientada aos Objetos, criámos um projeto que envolveu a elaboração de uma aplicação para gerir e reproduzir músicas, denominada de **SpotifUM**. A ideia era criar uma plataforma que simula algumas das funções básicas de um serviço de streaming musical, permitindo que os utilizadores ouçam músicas, criem playlists, vejam estatísticas e controlem o acesso de acordo com o seu tipo de assinatura.

Este projeto foi uma ótima oportunidade para colocar em prática e aprofundar conceitos essenciais de programação orientada a objetos, como encapsulamento, herança, polimorfismo e abstração.

Além disso, explorámos boas práticas de modelagem, como separar bem as responsabilidades entre classes, usar coleções para organizar melhor os dados e tratar exceções de forma segura. A aplicação foi criada de forma modular e fácil de expandir, incluindo funções como criar e editar playlists, gerir álbuns e músicas, distinguir diferentes tipos de utilizadores e gerar playlists personalizadas com base nos hábitos de escuta.

Neste relatório, descrevemos como foram tomadas as decisões de design, as funcionalidades que implementámos e os desafios que encontramos ao longo do caminho.

2. Arquitetura e Estrutura de Classes

Nesta secção é apresentada a estrutura interna da aplicação **SpotifUM**, destacando as principais classes que a compõem, os seus atributos e os métodos mais relevantes. O objetivo é fornecer uma visão clara da organização do sistema, bem como do papel que cada componente desempenha no funcionamento da aplicação. Para cada classe, são apresentadas as suas variáveis de instância, acompanhadas de uma breve explicação, e descritas as funcionalidades implementadas através dos métodos principais, excluindo os métodos habituais tais como *sets*, *gets*, *toString*, entre outros.

2.1. Classe Álbum

A classe **Álbum** representa uma coleção de músicas associada a um artista e a um título. É responsável por armazenar e gerir o conjunto de músicas pertencentes a um determinado álbum.

```
public class Album implements Serializable {
    private String titulo;
    private String artista;
    private List<Musica> musicas;

    public Album(String titulo, String artista) {
        this.titulo = titulo;
        this.artista = artista;
        this.musicas = new ArrayList<>();
    }
```

Figura 1 - Classe Album

Variáveis de Instância:

- **String título** – representa o título do álbum.
- **String artista** – indica o nome do artista ou banda responsável pelo álbum.
- **List<Musica> musicas** – lista de objetos **Musica** que compõem o álbum. É inicializada como uma lista vazia e protegida contra modificações externas através do método *Collections.unmodifiableList*.

Métodos principais:

- **adicionarMusica (Musica musica)** – Adiciona uma música ao álbum, desde que esta ainda não esteja presente na lista.
- **removerMusica (Musica musica)** – remove uma música específica da lista do álbum.
- **getMusicaPorNome (String nome)** – procura e devolve uma música do álbum com base no seu nome.
- **getDuracaoTotal()** – calcula e devolve a duração total do álbum, somando a duração de todas as músicas.

2.2. Classe Biblioteca

A classe **Biblioteca** funciona como um repositório da aplicação, armazenando os **álbuns** e as **playlists** existentes no sistema. É responsável pela gestão e organização destes elementos, permitindo adicioná-los e pesquisá-los de forma eficiente.

```
public class Biblioteca implements Serializable {
    private List<Album> albuns;
    private List<Playlist> playlists;

    public Biblioteca() {
        this.albuns = new ArrayList<>();
        this.playlists = new ArrayList<>();
    }
```

Figura 2 - Classe Biblioteca

Variáveis de Instância:

- **List<Album> albuns** – lista de objetos **Album**, representando todos os álbuns disponíveis na aplicação.
- **List<Playlist> playlists** – lista de objetos **Playlist**, representado as playlists criadas pelos utilizadores.

Métodos principais:

- **adicionarAlbum(Album album)** – adiciona um álbum à biblioteca, desde que ainda não exista.
- **adicionarPlaylist(Playlist playlist)** – adiciona uma playlist à biblioteca, garantindo que não haja duplicação.
- **procurarAlbum(String nome)** – procura um álbum na biblioteca com base no seu título.
- **procuraPlaylist(String nome)** – procura uma playlist pelo nome, aplicando o mesmo critério descrito acima.

2.3. Classe Estatísticas

A classe **Estatisticas** trata da análise dos dados que a aplicação vai acumulando, como utilizadores, músicas e playlists. O seu objetivo é fornecer informação útil e interessante, como quais as músicas mais ouvidas, quem são os utilizadores mais ativos ou qual o género musical mais popular. Esta classe permite dar uma visão geral sobre o uso da aplicação.

```
public class Estatisticas implements Serializable {

    private List<Utilizador> utilizadores;
    private List<Musica> musicas;
    private List<Playlist> playlists;

    public Estatisticas(List<Utilizador> utilizadores, List<Musica> musicas, List<Playlist> playlists) {
        this.utilizadores = utilizadores;
        this.musicas = musicas;
        this.playlists = playlists;
    }
```

Figura 3 - Classe Estatísticas

Variáveis de Instância:

- **List<Utilizador> utilizadores** – Lista todos os utilizadores registados na aplicação.
- **List <Musica musicas** – Lista todas as músicas disponíveis.
- **List<Playlist> playlists** – Lista de playlists presentes no sistema.

Métodos principais:

- **musicaMaisReproduzida()** – Devolve a música com maior número de reproduções.
- **interpreteMaisEscutado()** – Identifica o intérprete cuja soma total de reproduções é mais elevada.
- **utilizadorMaisAtivo()** – Determina o utilizador que mais músicas reproduziu através das suas playlists.
- **UtilizadorComMaisPontos()** – Retorna o utilizador com maior número de pontos acumulados.
- **generoMaisPopular()** – Calcula qual o género musical com mais reproduções globais, somando os contadores de reprodução por género.
- **numeroPlayListsPublicas()** – Devolve o número total de playlists públicas existentes.
- **utilizadorComMaisPlaylists()** – Identifica qual o utilizador que criou mais playlists.
- **utilizadorQueMaisOuviu()** – Identifica o utilizador que ouviu mais músicas no total, com base num contador próprio.

2.4. Classe Música

A classe **Musica** representa uma música individual na aplicação. Guarda toda a informação relevante sobre a mesma, como o nome, intérprete, género, duração, letra e

até uma versão textual da música. Além disso, permite simular a reprodução da música e regista quantas vezes foi ouvida, o que é útil para estatísticas ou recomendações futuras.

```java
public class Musica implements Serializable {
    private String nome;
    private String interprete;
    private String editora;
    private String letra;
    private List<String> musicaTexto;
    private Genero genero;
    private int duracao;
    private int contadorReproducao;

    public Musica(String nome, String interprete, String editora, String letra, List<String> musicaTexto,
                  Genero genero, int duracao) {
        this.nome = nome;
        this.interprete = interprete;
        this.editora = editora;
        this.letra = letra;
        this.musicaTexto = musicaTexto;
        this.genero = genero;
        this.duracao = duracao;
        this.contadorReproducao = 0;
    }
```

Figura 4 - Classe Musica

Variáveis de Instância:

- **String nome** - Representa o nome da música.
- **String interprete** – Identifica o nome do artista ou banda que interpreta a música.
- **String editora** – Indica o nome da editora associada à música.
- **String letra** – Representa a letra completa da música.
- **List<String> musicaTexto** - Representação textual da estrutura musical.
- **Genero género** - Género musical da música, definido por uma enumeração (Figura 7).
- **int duração** – Duração da música em segundos.
- **Int contadorReproducao** – Número de vezes que a música foi reproduzida.

Métodos Principais:

- **reproduzir()** – simula a reprodução da música, imprimindo a letra e o conteúdo textual no terminal. Incrementa automaticamente o contador de reproduções.
- **isExplicita()** – método que retorna **false** por defeito, podendo ser sobrescrito por subclasses que representam músicas com conteúdo explicito.

- **isMultimedia**() – retorna **false** por omissão. Este método pode ser reimplementado se a música tiver elementos multimédia associados.

2.4.1.Classe MusicaExplicita

A **MusicaExplicita** especializa a classe **Musica** para representar músicas que contém conteúdo explicito, mantendo toda a informação original e acrescentando a marcação de "explicita".

```
public class MusicaExplicita extends Musica {

    public MusicaExplicita(String nome, String interprete, String editora, String letra,
                           List<String> musicaTexto, Genero genero, int duracao) {
        super(nome, interprete, editora, letra, musicaTexto, genero, duracao);
    }
```

Figura 5 - Classe MusicaExplicita

Variáveis de Instância:

Não possui variáveis novas herda todas de **Musica** (nome, interprete, editora, letra, musicaTexto, género, duração e contadorReproducao).

Métodos Principais:

- **isExplicita()** – sobrescreve o método de base para devolver sempre **True**, indicando que a letra é explícita.

2.4.2.Classe MusicaMultimedia

A **MusicaMultimedia** estende **Musica** para incluir, além dos dados de uma música, o URL de um vídeo associado, de forma a distinguir músicas que têm componente audiovisual disponível.

```
public class MusicaMultimedia extends Musica {
    private String urlVideo;

    public MusicaMultimedia(String nome, String interprete, String editora, String letra,
                            List<String> musicaTexto, Genero genero, int duracao, String urlVideo) {
        super(nome, interprete, editora, letra, musicaTexto, genero, duracao);
        this.urlVideo = urlVideo;
    }
```

Figura 6 - Classe MusicaMultimedia

Variáveis de Instância:

- **String urlVideo** – endereço onde se encontra o vídeo da música.

As restantes variáveis de instância são herdades de **Musica**: nome, interprete, editora, letra, musicaTexto, género, duração e contadorReproducao.

Métodos Principais:

- **isMultimedia()** – Sobrescreve o método de base para devolver sempre **True**, sinalizando que esta música inclui conteúdo multimédia.

2.5. Enum Genero

O **Genero** define uma lista de géneros musicais padrão, permitindo categorizar cada música segundo o estilo ao qual pertence.

```
public enum Genero {
    Rock,
    Pop,
    Jazz,
    Blues,
    Classica,
    Eletronica,
    Hiphop,
    Reggae,
    Samba,
    Sertanejo,
    Funk,
    Metal,
    Country,
    Indie,
    Outro
}
```

Figura 7 - Enum Genero

Alguns dos estilos já definidos são: Rock, Pop, Jazz, Blues, Classica, Eletronica, HipHop, Reggae, Samba, Sertanejo, Funk, Metal, Country, Indie, Outro (refere-se a géneros não listados).

Este **Enum** é utilizado para atribuir de forma consistente o estilo de cada objeto **Musica**, ajudando na organização e filtragem por género.

2.6. Classe BuscadorDeMusicas

A classe **BuscadorDeMusicas** serve para filtrar e encontrar músicas a partir de uma lista predefinida, disponibilizando métodos que permitem pesquisar por género, duração, conteúdo explicito, interprete ou uma combinação desses critérios.

```
public class BuscadorDeMusicas {

    private List<Musica> todasAsMusicas;

    public BuscadorDeMusicas(List<Musica> musicas) {
        this.todasAsMusicas = musicas;
    }
```

Figura 8 - Classe BuscadorDeMusicas

Variáveis de Instância:

- **List<Musica> todasAsMusicas** – Lista todas as instâncias de Música sobre as quais se irão aplicar os filtros.

Métodos Principais:

- **buscarPorGenero(Genero genero)** – Retorna todas as músicas cujo género coincide com o argumento fornecido.
- **buscarPorDuracaoMaxima(int segundos)** – devolve as músicas cuja duração (em segundos) é menor ou igual ao valor indicado.
- **buscarExplicitas()** – Seleciona apenas músicas marcadas como explicitas.
- **buscarPorInterprete(String nome)** – Encontra todas as músicas cujo interprete coincida com o nome dado.
- **buscarPorMultiplosFiltros(Genero género, int duracaoMaxima, boolean explicita)** – Aplica em simultâneo os filtros de género (se não for *null*), duração máxima (se for maior do que zero) e conteúdo explicito (se for *true*), devolvendo apenas músicas que satisfazem todos os critérios.

2.7. Interface PlanoSubscricao

A interface **PlanoSubscricao** define um contrato para diferentes tipos de subscrição, especificando como calcular pontos de utilizador e que funcionalidades cada plano disponibiliza (criação de playlists e lista de favoritos).

```java
public interface PlanoSubscricao {
    int calcularPontos(int pontosAtuais);
    boolean podeCriarPlaylists();
    boolean podeTerFavoritos();
}
```

Figura 9 - Interface PlanoSubscricao

Variáveis de Instância:

Não possui variáveis de instância, uma vez que se trata de uma interface que apenas define métodos a implementar.

Métodos principais:

- **calcularPontos(int PontosAtuais)** – Recebe os pontos atuais do utilizador e devolve o novo total consoante as regras do plano.
- **podeCriarPlaylists()** – Indica se o plano permite ao utilizador criar playlists.
- **podeTerFavoritso()** – Indica se o plano permite ao utilizador assinalar músicas como favoritas.

Desta forma, ao utilizar esta interface, assegura-se a consistências na implementação das operações de cálculo de pontos e na validação das permissões para criação de playlists e gestão de favoritos, independentemente do plano de subscrição.

2.8. Classe PremiumBase

A classe **PremiumBase** representa um plano de subscrição de nível intermédio, implementando o contrato definido pela interface **PlanoSubscrição**.

```java
public class PremiumBase implements PlanoSubscricao, Serializable {

    @Override
    public int calcularPontos(int pontosAtuais) {
        return pontosAtuais + 10;
    }

    @Override
    public boolean podeCriarPlaylists() {
        return true;
    }

    @Override
    public boolean podeTerFavoritos() {
        return false;
    }
}
```

Figura 10 - Classe PremiumBase

Variáveis de instância:

Não possui variáveis de instância, uma vez que todas as funcionalidades são concretizadas pelos métodos definidos nas interfaces implementadas.

Métodos Principais:

- **calcularPontos(int pontosAtuais)** – Acrescenta um bónus fixo de 10 pontos ao total atual, favorecendo assim os utilizadores que subscrevam este plano.
- **podeCriarPlaylists()** – Retorna ***True***, indicando que este plano autoriza a criação de novas playlists.
- **podeTerFavoritos()** – Retorna ***False***, indicando que a gestão de favoritos não está disponível neste nível de subscrição.

Desta forma, a classe **PremiumBase** assegura a aplicação consistente das regras de subscrição definidas pela interface **PlanoSubscrição**.

2.9. Classe PremiumTop

A classe **PremiumTop** representa o nível mais elevado de subscrição, implementando o cotrato definido pela interface **PlanoSubscrição**.

```java
public class PremiumTop implements PlanoSubscricao, Serializable {

    @Override
    public int calcularPontos(int pontosAtuais) {
        return pontosAtuais + (int)(pontosAtuais * 0.025); // +2.5%
    }

    @Override
    public boolean podeCriarPlaylists() {
        return true;
    }

    @Override
    public boolean podeTerFavoritos() {
        return true;
    }
}
```

Figura 11 - Classe PremiumTop

Variáveis de Instância:

Não possui variáveis de instância, uma vez que toda a lógica se encontra nos métodos herdados das interfaces implementadas.

Métodos Principais:

- **calcularPontos(int pontosAtuais)** – Aplica um bónus de 2.5% sobre os pontos existentes, convertendo o resultado para inteiro, e devolve o total atualizado.
- **podeCriarPlaylists()** – Devolve ***True***, permitindo ao utilizador criar playlists sem limitações.
- **podeTerFavoritos()** – Devolve ***True***, autorizando o utilizador a assinalar as músicas como favoritas.

Desta maneira, a classe **PremiumTop** garante que as regras dos pontos e permissões definidas em **PlanoSubscricao** foram aplicadas.

2.10. Classe Utilizador

A Classe **Utilizador** define um utilizador da aplicação. É abstrata, ou seja, serve como base para diferentes tipos de utilizadores. Cada utilizador tem uma biblioteca pessoal onde pode guardar álbuns e playlists, acumula pontos e regista quantas músicas ouviu.

```
public abstract class Utilizador implements Serializable {
    protected String nome;
    protected String email;
    protected String morada;
    protected int pontos;
    protected int musicasOuvidas;
    protected Biblioteca biblioteca = new Biblioteca();

    public Utilizador(String nome, String email, String morada) {
        this.nome = nome;
        this.email = email;
        this.morada = morada;
        this.pontos = 0;
        this.musicasOuvidas = 0;
    }
```

Figura 12 - Classe Utilizador

Variáveis de Instância:

- **String nome** - Representa o nome do utilizador.
- **String email** - Indica o email do utilizador
- **String morada** - Corresponde à morada do utilizador
- **int pontos** - Descreve os pontos acumulados com interações na aplicação (ouvir músicas)
- **int musicasOuvidos** - Representa o total de músicas que o utilizador ouviu até ao momento
- **Biblioteca biblioteca** – biblioteca pessoal do utilizador, onde guarda os seus álbuns e playlists.

Métodos principais:

- **adicionarAlbum (Album álbum)** – Adiciona um álbum á biblioteca pessoal do utilizador
- **adicionarPlaylist (Playlist playlist)** – Adiciona uma playlist à biblioteca pessoal do utilizador

- **reproduzirMusica (Musica musica)** – Método abstrato que define a forma como o utilizador reproduz uma música (implementado nas subclasses)

2.10.1. Classe Desenvolvedor

Esta classe representa um tipo especial de utilizador com permissões adicionar, nomeadamente a capacidade de criar álbuns e músicas (normais ou explicitas). Um desenvolvedor pode reproduzir músicas, mas não acumula pontos com isso. Esta classe herda da classe abstrata **Utilizador**, aproveitando a sua estrutura base de nome, email, morada, pontos e biblioteca.

```
public class Desenvolvedor extends Utilizador {

    public Desenvolvedor(String nome, String email, String morada) {
        super(nome, email, morada);
    }
```

Figura 13 - Classe Desenvolvedor

Variáveis de instância:

Esta classe herda todas as variáveis de instância da classe **Utilizador**.

Métodos principais:

- **reproduzirMusica (Musica musica)** – Reproduz a música recebida, incrementando o número de músicas ouvidas, mas sem atribuir pontos.
- **criarAlbum (String título, String artista)** – Cria um álbum com o título e artista fornecidos)
- **criarMusica (String nome, String interprete, String editora, String letra, List<String> texto, Genero género, int duracao)** – Cria uma música normal com os parâmetros dados.
- **criarMusicaExplicita (String nome, String interprete, String editora, String letra, List<String> texto, Genero género, int duracao)** – Cria uma música marcada como explicita, utilizador a subclasse MusicaExplicita.

2.10.2. Classe UtilizadorFree

Esta classe representa um utilizador com acesso gratuito à aplicação. Tal como o nome indica, um utilizador *free* pode ouvir músicas normalmente, mas com algumas limitações que o distinguem de utilizadores premium. A cada reprodução de uma música, este tipo de utilizador acumula pontos.

```
public class UtilizadorFree extends Utilizador {

    public UtilizadorFree(String nome, String email, String morada) {
        super(nome, email, morada);
    }
```

Figura 14 - Classe UtilizadorFree

Variáveis de Instância:

Esta classe herda todas as variáveis de instância da classe **Utilizador**.

Métodos principais:

- **reproduzirMusica (Musica musica)** – Reproduz uma música indicada, incrementa o contador de músicas ouvidas e adiciona 5 pontos ao total do utilizador.

2.10.3. Classe UtilizadorPremium

Esta classe representa um utilizador com subscrição *premium*, permitindo-lhe acesso a funcionalidades adicionais da aplicação (como criação de playlists ou visualização de favoritos), dependendo do plano de subscrição. Ao reproduzir músicas, a lógica de atribuição de pontos é delegada ao plano escolhido. Se o plano for do tipo ***PremiumTop***, o utilizador inicia automaticamente com 100 pontos.

```
public class UtilizadorPremium extends Utilizador {
    private PlanoSubscricao plano;

    public UtilizadorPremium(String nome, String email, String morada, PlanoSubscricao plano) {
        super(nome, email, morada);
        this.plano = plano;
            if (plano instanceof PremiumTop) {
                this.pontos = 100;
            }

    }
```

Figura 15 - Classe UtilizadorPremium

Variáveis de Instância:

- **PlanoSuscricao plano** – Representa o plano de subscrição associado ao utilizador, o qual define comportamentos como pontuação ou permissões específicas.

Métodos Principais:

- **reproduzirMusica (Musica musica)** – Reproduz a música indicada, incrementa o número de músicas ouvidas e atualiza os pontos com base na lógica do plano de subscrição.
- **podeCriarPlaylists()** – Retorna se o utilizador pode criar playlists, conforme definido pelo plano.
- **podeVerFavoritos()** – Retorna se o utilizador pode visualizar os favoritos, conforme o permitido pelo plano.

2.11. Classe Playlist

A classe **Playlist** é uma classe abstrata que representa uma coleção de músicas agrupadas com um nome, com a possibilidade de serem reproduzidas em sequência ou de forma aleatória. Esta classe implementa a interface **Serializable**, permitindo que o estado dos seus objetos seja guardado em ficheiro para posterior reutilização.

Trata-se de uma estrutura base sobre a qual outras playlists específicas poderão ser construídas.

```
public abstract class Playlist implements Serializable {
    protected String nome;
    protected List<Musica> musicas;
    protected boolean modoAleatorio;
    private boolean publica;

    public Playlist(String nome, boolean publica) {
        this.nome = nome;
        this.musicas = new ArrayList<>();
        this.modoAleatorio = false;
        this.publica = publica;
    }
```

Figura 16 - Classe Playlist

Variáveis de Instância:

- **String nome** – Representa o nome da playlist.
- **List<Musica> musicas** – Lista de objetos do tipo **Musica**, que contém as músicas adicionadas à playlist.
- **Boolean modoAleatorio** – Indica se a reprodução das músicas deve seguir uma ordem aleatória ou não.
- **Boolean publica** – Indica se a playlist é pública ou privada

Métodos Principais:

- **adicionarMusica (Musica musica)** – Adiciona uma musica à playlist, desde que esta ainda não existe na lista. Garante unicidade, ou seja, evita a duplicação de músicas.
- **removerMusica (Musica musica)** – Remove uma música da playlist, caso exista. Se a música não estiver presente, o método termina sem qualquer efeito.
- **reproduzir()** – Responsável por iniciar a reprodução da playlist. Se o modo aleatório estiver ativado, embaralha a ordem das músicas antes da reprodução. Cada música é reproduzida utilizado o método **reproduzir()** da própria classe **Musica**, mostrando o nome, intérprete, letra e representação textual.
- **reproduzirComControlo (Utilizador utilizado)** – Permite a reprodução controlada da playlist, criando um ***Thread*** que simula a execução das músicas com tempo e letra a ser apresentada palavra a palavra. Este método só permite controlo manual (com os comandos "next", "prev" e "stop") para utilizadores que não sejam do tipo **UtilizadorFree**. Caso o utilizador seja ***Free***, a reprodução ocorre normalmente, mas sem controlo. O Controlo da reprodução é feito através de inputs no terminal e permite alterar dinamicamente a música a tocar ou terminal a reprodução.

2.11.1. Classe PlaylistExplicita

A classe **PlaylistExplicita** é uma subclasse concreta da classe abstrata **Playlist**, especializada na criação automática de playlists que contém exclusivamente músicas marcadas como explicitas.

Esta classe tem como principal objetivo filtrar e agrupar apenas as músicas com conteúdo explicito, assumindo que esse conteúdo foi previamente assinalado nos objetos Musica através do método **isExplicita()**.

```
public class PlaylistExplicita extends Playlist {

    public PlaylistExplicita(String nome, boolean publica, List<Musica> candidatas) {
        super(nome, publica);
        for (Musica m : candidatas) {
            if (m.isExplicita()) {
                musicas.add(m);
            }
        }
    }
```

Figura 17 - Classe PlaylistExplicita

Método Principal:

- **PlaylistExplicita(String nome, boolean publica, List<Musica> candidatas)** – No momento da criação da playlist, é fornecida uma lista de músicas candidatas. O construtor percorre esta lista e adiciona apenas as músicas que devolvem **true** no método **isExplicita()**. Desta forma, garante-se que a playlist final é composta exclusivamente por músicas explicitas, sem necessidade de filtragem posterior por parte do utilizador.

2.11.2. Classe PlaylistFavoritos

A classe **PlaylistFavoritos** é uma subclasse concreta da classe abstrata **Playlist** e representa uma playlist gerada automaticamente a partir de músicas marcadas como favoritas por um utilizador.

O principal objetivo da classe é agregar rapidamente um conjunto de músicas favoritas, facilitando ao utilizador o acesso direto à sua seleção pessoal de músicas mais apreciadas.

```
public class PlaylistFavoritos extends Playlist {

    public PlaylistFavoritos(String nome, boolean publica, List<Musica> favoritas) {
        super(nome, publica);
        this.musicas.addAll(favoritas);
    }
```

Figura 18 - PlaylistFavoritos

Método Principal:

- **PlaylistFavoritos(String nome, boolean publica, List<Musica> favoritas** – Este construtor recebe como parâmetro uma lista de músicas previamente identificadas como favoritas. Ao instanciar o objeto, dessa lista é imediatamente adicionada à playlist, sem validações adicionais, assumindo que a seleção já foi devidamente filtrada antes.

2.11.3. Classe PlaylistNormal

A classe **PlaylistNormal** representa uma playlist personalizada criada manualmente por um utilizador. Esta playlist não aplica quaisquer critérios automáticos de filtragem de músicas, ao contrário de outras classes como a **PlaylistExplicita** ou a **PlaylistFavoritos**.

Cada **PlaylistNormal** é criada com uma visibilidade privada por omissão e contém um autor associado, permitindo identificar quem foi o responsável pela sua criação.

```
public class PlaylistNormal extends Playlist {

    private String autor;

    public PlaylistNormal(String nome, String autor) {
        super(nome, publica:false);
        this.autor = autor;
    }
```

Figura 19 - PlaylistNormal

Variáveis de instância:

- **String autor** – Guarda o nome do utilizador que criou a playlist.

Herda também da sua superclasse (**Playlist**) as variáveis: nome, musicas, modoAleatorio e publica.

Métodos principais:

Apenas contém o construtor da playlist com o nome e autor fornecidos.

2.11.4. Classe PlaylistPorDuracao

A classe **PlaylistPorDuracao** é uma especialização da classe abstrata **Playlist** e é usada para criar playlists com base num conjunto de músicas filtradas, as quais podem ser selecionadas com base na duração de cada uma. Assim, esta classe permite criar playlists onde as músicas são selecionadas considerando a sua duração. Como as outras playlists, a **PlaylistPorDuracao** pode ser pública ou privada.

```
public class PlaylistPorDuracao extends Playlist {

    public PlaylistPorDuracao(String nome, boolean publica, List<Musica> musicasFiltradas) {
        super(nome, publica);
        this.musicas.addAll(musicasFiltradas);
    }
```

Figura 20 - PlaylistPorDuracao

Variáveis de Instância:

Não possui variáveis de instância adicionais às herdadas da sua superclasse **Playlist**.

Métodos Principais:

- **PlaylistPorDuracao (String nome, boolean publica, List<Musica> musicasFiltradas)** – Constrói uma playlist com base nas músicas filtradas passadas como argumento. Essas músicas são adicionadas à playlist ao ser criada.

2.11.5. Classe PlaylistPublica

A classe **PlaylistPublica** é uma especializada da classe abstrata **Playlist**, sendo utilizada para representar playlists que são públicas e associadas a um autor especifico. Ou seja, esta classe permite criar playlists que podem ser visualizadas por outros utilizadores. O

autor é indicado no momento da criação da playlist e a visibilidade da playlist é definida como ***true*** para indicar que é pública.

```
public class PlaylistPublica extends Playlist {
    private String autor;

    public PlaylistPublica(String nome, String autor) {
        super(nome, publica:true);
        this.autor = autor;
    }
```

Figura 21 - Classe PlaylistPublica

Variáveis de Instância:

- **String autor** – Armazena o nome do utilizador que criou a playlist pública.

Métodos principais:

- **PlaylistPublica (String nome, String autor)** – Constrói uma playlist pública, atribuindo-lhe um nome e associando-a a um autor específico. A playlist é configurada para ser pública através do parâmetro ***true*** passado para o construtor da classe **Playlist**.

2.12. Classe SpotifUM

A Classe **SpotifUM** representa o núcleo da aplicação, centralizando a gestão de utilizadores, músicas, álbuns e playlists. Permite adicionar e procurar cada um destes elementos, assim como gerar estatísticas globais do sistema. Também disponibiliza métodos para guardar e carregar o estado da aplicação usando serialização.

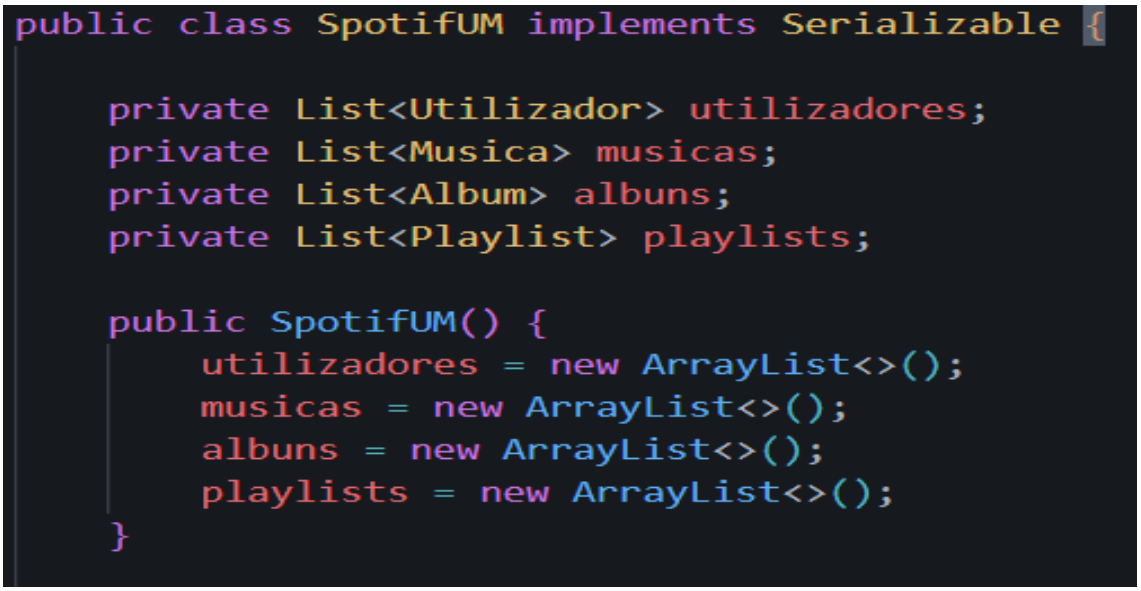

Figura 22 - Classe SpotifUM

Variáveis de instância:

- **List<Utilizador utilizadores** – Lista de todos os utilizadores registados na aplicação.
- **List<Musica> musicas** – Lista de todas as músicas disponíveis na aplicação.
- **List<Album> albuns** – Lista de álbuns registados no sistema.
- **List<Playlist> playlists** – Lista de playlists criadas no sistema (por utilizadores ou geradas automaticamente).

Métodos Principais:

- **adicionarUtilizador (Utilizador u)** – Adiciona um novo utilizador à aplicação.
- **procurarUtilizador (String nome)** – Procura um utilizador pelo nome (ignora maiúsculas/minúsculas).
- **adicionarMusica (Musica m)** – Adiciona uma nova música à base de dados da aplicação.
- **adicionarAlbum (Album álbum)** – Regista um novo álbum no sistema.
- **procurarAlbum (String titulo)** – Procura um álbum com o titulo especificado.
- **adicionarPlaylist (Playlist p)** – Adiciona uma nova playlist ao sistema.
- **procurarPlaylist (String nome)** – Procura uma playlist com o nome indicado.
- **gerarEstatisticas()** – Gera estatísticas globais a partir das listas de utilizador, músicas e playlists.

- **guardarEstado (String ficheiro)** – Guarda o estado atual da aplicação num ficheiro, através de serialização.
- **carregarEstado (String ficheiro)** – Carrega o estado da aplicação a partir dum ficheiro.

2.13. Classe MenuSpotifUM

A classe **MenuSpotifUM** representa o ponto de entrada da aplicação SpotifUM e é responsável por toda a interação com o utilizador. Através de um sistema de menus, permite aceder às principais funcionalidades: criação e login de utilizadores, reprodução de músicas, álbuns e playlists, bem como a geração de playlists com filtros personalizados.

No momento da inicialização, a classe tenta carregar o estado anterior da aplicação a partir de um ficheiro. Caso não seja possível, é criada uma instância limpa, garantindo a continuidade do funcionamento, O método principal **executa()** implementa o ciclo de vida da aplicação, apresentado o menu inicial e respondendo às escolhas do utilizador.

A classe também permite diferencial a experiência entre utilizadores casuais e premium, apresentado menus adaptados conforme o tipo de utilizador. A reprodução de conteúdos é feita com base no nome introduzido, e a criação de playlists com filtros recorre a mecanismos de pesquisa para gerar listas personalizadas com base em critérios como género, artista ou ano.

Ao terminar a sessão, o estado da aplicação pode ser guardado, assegurando a persistência dos dados. Assim, **MenuSpotifUM** atua como a ponte entre o utilizador e a lógica interna da aplicação, garantindo uma interação fluida, organizada e fiável.

2.14. Classe Auxiliar GerarDadosMeusInicias

Embora esta classe não faça parte da lógica principal da aplicação, foi criada com o objetivo de facilitar a geração de dados e teste e demonstração. Esta classe contém o método **main**, responsável por instanciar vários objetos do sistema (músicas, álbuns, utilizadores, playlists) e preencher o sistema **SpotifUM** com conteúdo significado, pronto a ser usado ou avaliado.

A sua principal função é permitir testar e demonstrar as funcionalidades da aplicação com um conjunto coerente e diversificado de dados, sem necessidade de inserção manual.

Esta classe utiliza diretamente os métodos públicos das outras entidades e termina com a gravação do estado da aplicação para ficheiro, o que permite carregar os dados numa sessão posterior.

Nota: Algumas classes implementam a interface **Serializable** com o objetivo de permitir a sua persistência em ficheiro. Esta funcionalidade é essencial para guardar o estado do sistema entre sessões, garantindo que os dados associados aos objetos (como utilizadores, playlists ou músicas) possam ser escritos e lidos de forma eficiente. Desta forma, é possível assegurar a continuidade da informação sem necessidade de reconstrução manual.

3. Diagrama de Classes

O Desenvolvido da aplicação **SpotifUM** seguiu os princípios da Programação Orientada aos Objetos, com especial atenção à organização e reutilização de código. O diagrama de classes apresentado neste seção reflete a estrutura da solução implementada, representado as principais entidades do sistema, as suas relações e os comportamentos associados.

Este diagrama permite visualizar a forma como as classes foram organizadas e como se articulam entre si para dar resposta aos requisitos funcionais definidos, nomeadamente a gestão de músicas, playlists, utilizadores e planos de subscrição.

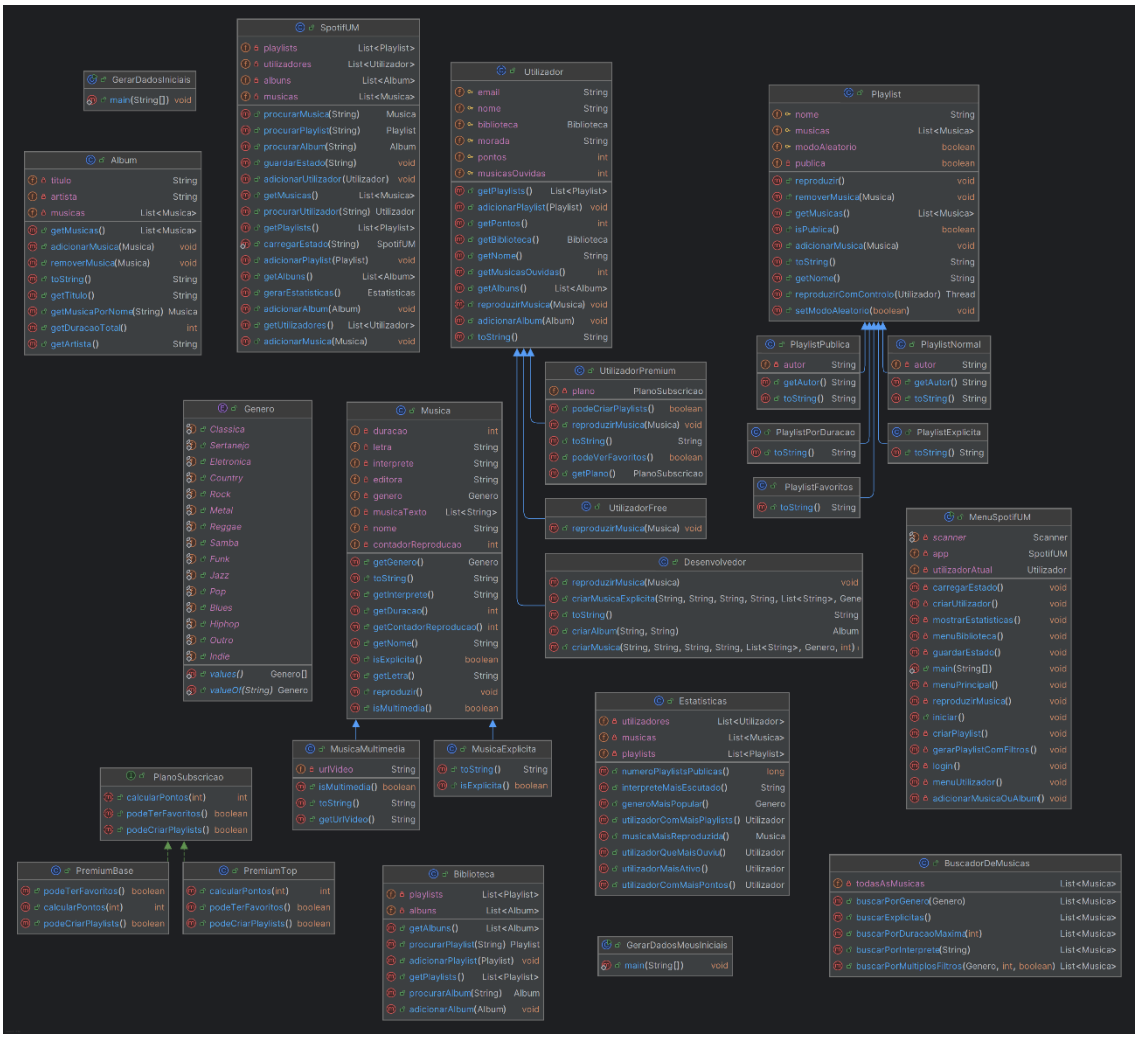

Figura 23 - Diagrama de Classes

4. Descrição de Funcionalidades

A aplicação **SpotifUM** implementa um conjunto de funcionalidades que permitem a gestão de músicas, utilizadores e playlists, bem como a reprodução de conteúdos e geração de estatísticas. Abaixo apresenta-se a lista de funcionalidades principais desenvolvidas, com demonstração através de figuras.

1. Gestão de Entidades
    a. Criar utilizadores, com diferentes tipos de plano (Free, PremiumBase, PremiumTop).

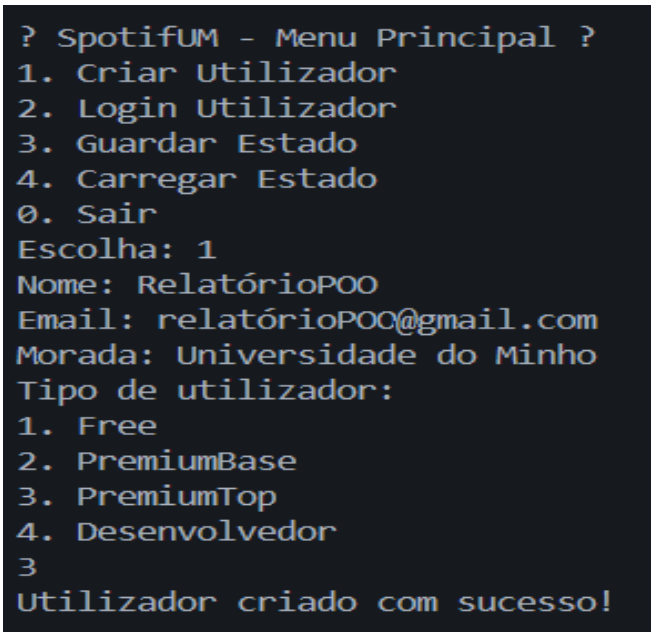

Figura 24 - Criação de Utilizador PremiumTop

Para efeitos de demonstração, foi criado apenas um utilizador com o plano *PremiumTop*. No entanto, como se pode verificar na Figura 23, é possível criar utilizadores com qualquer um dos planos disponíveis na aplicação.

b. Criar álbuns, que agrupam músicas

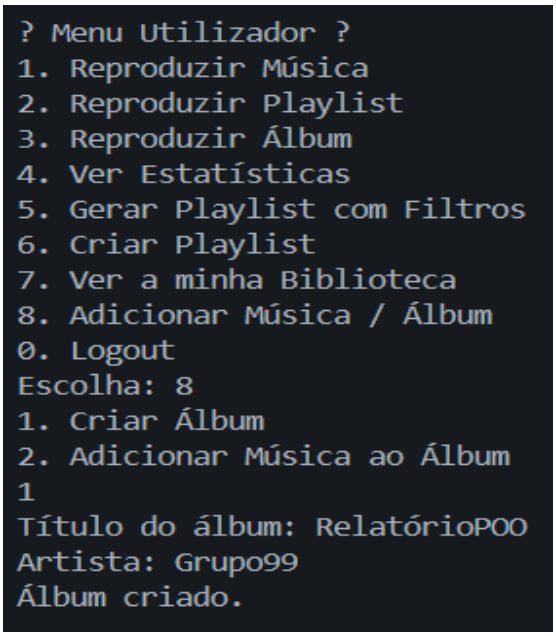

Figura 25 - Criação de Álbuns

c. Criar músicas e adicioná-la a um álbum à escolha.

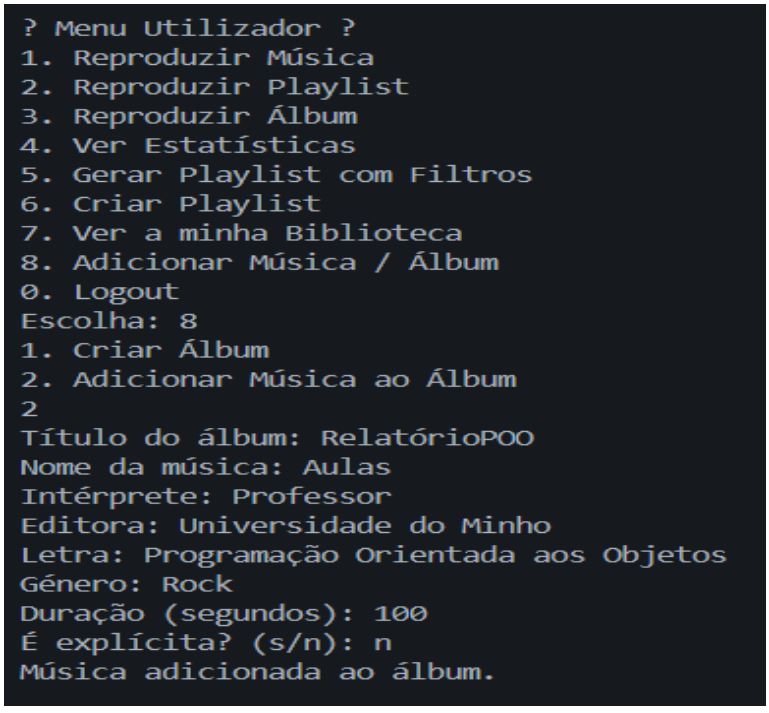

Figura 26 - Criar Música e adicioná-la a um álbum

Para permitir a criação de álbuns e músicas, foi implementada a funcionalidade de registo como desenvolvedor. Apenas utilizadores com este tipo de registo podem adicionar novos álbuns e músicas, uma vez que o grupo considerou inadequado permitir que utilizadores comuns criassem conteúdo musical.

d. Criar playlists, associadas a utilizadores premium.

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
0. Logout
Escolha: 6
Nome da nova playlist: PlayListRelatórioPOO
A playlist será pública? (sim/nao): sim
Escolha as músicas para a playlist (digite o nome da música ou 'fim' para terminar):
Nome da música: Agiota
Nome da música: Vivo
Nome da música: fim
? Playlist "PlayListRelat?rioPOO" criada.
```

Figura 27 - Criação de Playlist

2. Reprodução de Conteúdos
    a. Reproduzir músicas individuais, mostrando a letra no ecrã.

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
0. Logout
Escolha: 1
Nome da música: Agiota
Reproduzindo: Agiota por Dillaz
? Letra: o seu filho anda a ver se se queima
DO RE MI FA
SO LA SI DO
Música reproduzida.
```

Figura 28 - Reprodução de Música

    b. Reproduzir álbuns completos.

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
0. Logout
Escolha: 3
? Álbuns disponíveis:
1. O Proprio
Escolhe o número do álbum para reproduzir: 1
? Reproduzindo playlist com controlo: O Proprio
Reproduzindo: Gangta por Dillaz
? Letra: Gangsta, 'tou no sítio do costume
DO RE MI FA
SO LA SI DO
? Tocando: Gangta [216s]
? Letra: Gangsta, 'tou no sítio do costume
Reproduzindo: Agiota por Dillaz
? Letra: o seu filho anda a ver se se queima
DO RE MI FA
SO LA SI DO
? Tocando: Agiota [165s]
? Letra: o seu filho anda a ver se se queima
```

Figura 29 - Reprodução de Álbum

Para a funcionalidade de reprodução de álbuns, foram utilizadas *Threads*, após consulta e respetiva autorização concedida por um dos docentes responsáveis pelas aulas práticas da unidade curricular. Importa referir que, para que um álbum possa ser reproduzido, é necessário que este tenha sido previamente adicionado à biblioteca do utilizador.

c. Reproduzir playlists públicas ou privadas, com controlo da ordem de reprodução

```
Qual o nome da playlist?
Random Guga
? Reproduzindo playlist com controlo: Random Guga
Reproduzindo: Agiota por Dillaz
? Letra: o seu filho anda a ver se se queima
DO RE MI FA
SO LA SI DO
? Tocando: Agiota [165s]
? Letra: o seu filho anda a ver se se queima
next
Reproduzindo: Nota 100 por Dillaz
? Letra: Ela é uma obra-prima, não sei quem a fez
DO RE MI FA
SO LA SI DO
? Tocando: Nota 100 [168s]
? Letra: Ela é uma obra-prima, não sei quem a fez
next
Reproduzindo: Fica So. por Van Zee
? Letra: Fica Só.
DO RE MI FA
SO LA SI DO
? Tocando: Fica So. [197s]
? Letra: Fica Só.
prev
Reproduzindo: Fica So. por Van Zee
? Letra: Fica Só.
DO RE MI FA
SO LA SI DO
? Tocando: Fica So. [197s]
? Letra: Fica Só.
stop
? Reprodução terminada.
```

Figura 30 - Reprodução duma playlist pública com controlo da ordem de reprodução

d. Reproduzir playlists para utilizadores Free.

```
? SpotifUM - Menu Principal ?
1. Criar Utilizador
2. Login Utilizador
3. Guardar Estado
4. Carregar Estado
0. Sair
Escolha: 2
Nome de utilizador: RelatórioPOOFree
Login feito como: Relat?rioPOOFree (Relat?rioPOOFree@gmail.com) - 0 pontos

? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
7. Ver a minha Biblioteca
0. Logout
Escolha: 2
Qual o nome da playlist?
Random Guga
? Reproduzindo playlist com controlo: Random Guga
?? Utilizadores Free não podem controlar a reprodução.
Reproduzindo: Agiota por Dillaz
? Letra: o seu filho anda a ver se se queima
DO RE MI FA
SO LA SI DO
? Tocando: Agiota [165s]
? Letra: o
```

Figura 31 - Reprodução de playlists para utilizadores Free

3. Atribuição de Pontos
    a. Atribuição automática de pontos a cada utilizador consoante o tipo de plano e o número de músicas reproduzidas.

```
? SpotifUM - Menu Principal ?
1. Criar Utilizador
2. Login Utilizador
3. Guardar Estado
0. Sair
Escolha: 2
Nome de utilizador: RelatórioPOO
Login feito como: Relat?rioPOO (relat?rioPOO@gmail.com) - 100 pontos [Premium]
```

Figura 32 - Atribuição automática de pontos para utilizador PremiumTop

Mais uma vez, a demonstração da atribuição de pontos foi realizada com um utilizador do tipo *PremiumTop*, sendo que o funcionamento é semelhante para os restantes tipos de plano.

4. Estatísticas do Sistema
    a. Identificar a música mais reproduzida.
    b. Identificar o intérprete mais escutado.
    c. Indicar qual o utilizador com mais pontos.
    d. Apresentar o género musical mais reproduzido.
    e. Contar quantas playlists públicas existem.
    f. Indicar qual o utilizador que tem mais playlists.
    g. Mostrar o utilizador que mais músicas ouviu.

```
? Estatísticas:
Música mais reproduzida: Agiota - Dillaz [Hiphop] (165s)
Intérprete mais escutado: Dillaz
Utilizador com mais pontos: Relat?rioPOO (relat?rioPOO@gmail.com) - 116 pontos [Premium]
Género mais popular: Hiphop
Nº de playlists públicas: 1
Utilizador com mais playlists: Dinis (dinis@gmail.com) - 0 pontos [Premium]
?Utilizador que mais músicas ouviu: Relat?rioPOO (relat?rioPOO@gmail.com) - 116 pontos [Premium] com 8 músicas ouvidas.
```

Figura 33 - Estatísticas do Sistema

5. Músicas com Características Especiais
    a. Criar músicas explicitas

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
8. Adicionar Música / Álbum
0. Logout
Escolha: 8
1. Criar Álbum
2. Adicionar Música ao Álbum
1
Título do álbum: Explicitas
Artista: Vários
Álbum criado.

? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
8. Adicionar Música / Álbum
0. Logout
Escolha: 8
1. Criar Álbum
2. Adicionar Música ao Álbum
2
Título do álbum: Explicitas
Nome da música: Sicko Mode
Intérprete: TravisScott
Editora: Epic
Letra: I did half a Xan, 13 hours 'til I land, had me out like a light
Género: Hiphop
Duração (segundos): 100
É explícita? (s/n): s
Música adicionada ao álbum.
```

Figura 34 - Criação de Músicas Explicitas

6. Geração Automática de Playlists
    a. Gerar playlist com as preferências musicais de um utilizador premium.
    b. Gerar playlist com tempo máximo definido e preferências do utilizador.

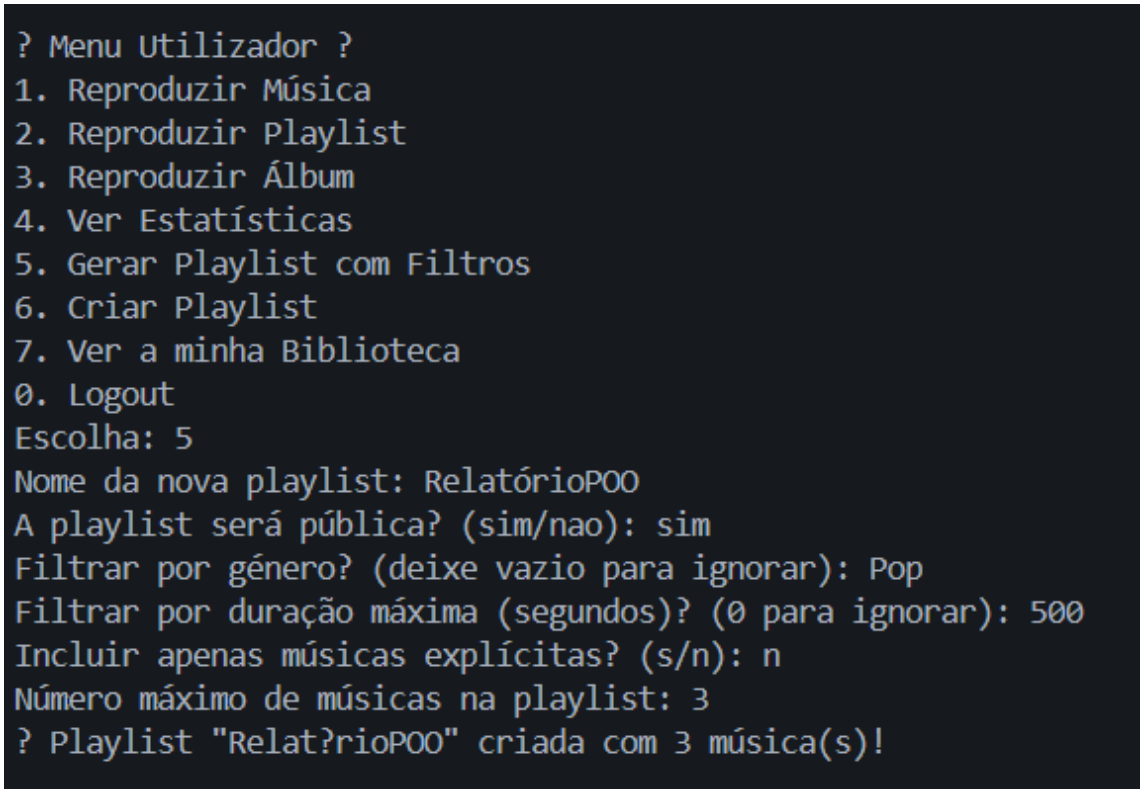

Figura 35 - Geração Automática de Playlists

c. Gerar playlist com apenas músicas explicitas, baseando-a nos gostos do utilizador

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
8. Adicionar Música / Álbum
0. Logout
Escolha: 5
Nome da nova playlist: MusicasExplicitas
A playlist será pública? (sim/nao): nao
Filtrar por género? (deixe vazio para ignorar):
Filtrar por duração máxima (segundos)? (0 para ignorar): 250
Incluir apenas músicas explícitas? (s/n): s
Número máximo de músicas na playlist: 2
? Playlist "MusicasExplicitas" criada com 2 música(s)!
```

Figura 36 - Geração de Playlist com músicas explicitas

7. Salvaguarda e Recuperação do Estado
    a. Guardar o estado da aplicação num ficheiro.

```
? Menu Utilizador ?
1. Reproduzir Música
2. Reproduzir Playlist
3. Reproduzir Álbum
4. Ver Estatísticas
5. Gerar Playlist com Filtros
6. Criar Playlist
7. Ver a minha Biblioteca
0. Logout
Escolha: 0

? SpotifUM - Menu Principal ?
1. Criar Utilizador
2. Login Utilizador
3. Guardar Estado
4. Carregar Estado
0. Sair
Escolha: 3
Estado guardado com sucesso.
```

Figura 37 - Gravação dum estado do sistema

b. Carregar o estado guardado numa nova execução.

```
? SpotifUM - Menu Principal ?
1. Criar Utilizador
2. Login Utilizador
3. Guardar Estado
4. Carregar Estado
0. Sair
Escolha: 4
Estado carregado com sucesso.

? SpotifUM - Menu Principal ?
1. Criar Utilizador
2. Login Utilizador
3. Guardar Estado
4. Carregar Estado
0. Sair
Escolha: 2
Nome de utilizador: RelatórioPOO
Login feito como: Relat?rioPOO (Relat?rioPOO@gmail.com) - 135 pontos [Premium]
```

Figura 38 - Leitura dum estado do Sistema

5. Conclusão

O desenvolvimento da aplicação **SpotifUM** foi um projeto que permitiu colocar em prática os princípios básicos da Programação Orientada a Objetos de uma forma bastante prática. Isso ajudou bastante a consolidar conceitos como encapsulamento, herança, polimorfismo e modularidade. Durante o projeto, conseguimos criar um sistema funcional que simula uma plataforma de streaming musical, diferenciando tipos de utilizadores, criando e manipulando playlists, além de gerar estatísticas personalizadas.

A arquitetura que construímos mostrou-se bastante eficiente para suportar as funcionalidades planeadas e, ao mesmo tempo, foi pensada para ser flexível o suficiente para permitir futuras melhorias. Ainda assim, percebemos algumas áreas que podem ser melhoradas. Por exemplo, trocar algumas estruturas de dados mais simples por *HashMaps* poderia tornar o acesso e a procura por elementos mais rápidos, especialmente em operações que acontecem com frequência, como procurar músicas, álbuns ou usuários. Além disso, implementar um sistema de **gravação automática do estado da aplicação em arquivo** — como guardar após cada alteração importante — ajudaria a deixar o sistema mais robusto e facilitar a vida do utilizador, evitando que se percam dados que não foram guardados manualmente.

No geral, podemos dizer que o projeto conseguiu atingir seus principais objetivos pedagógicos e funcionais, e serve como uma base sólida para desenvolver aplicações mais avançadas no futuro.