Desenvolvimento Centrado em Objetos 2022/2023
Projeto 1 - LeiTunes

58180 - Rodrigo Correia
58188 - Laura Cunha

1 Decisões tomadas:

1. **Classe *MusicLibrary*:**

   Nesta classe escolhemos estender a classe *AbsSubject<SongLibraryEvent>* em vez da classe *AbsQListWithSelection<Song>*, isto porque não possuímos nenhuma implementação concreta da *AbsSubject*, logo se a quiséssemos implementar e usar por composição teríamos de criar uma classe concreta que implementasse a abstrata, e aí sim, utiliza-la com atributo. Também não faria muito sentido haver uma classe de subject externa à MusicLibrary, usada apenas para chamar métodos já implementados na abstrata. Em contrapartida, a *AbsQListWithSelection*, já apresenta uma classe concreta implementada (a *ArrayQListWithSelection*), esta representa uma boa estrutura de dados para guardar as Songs na *MusicLibrary*, com isso, implementámos os métodos da interface *QListWithSelection* através de composição e forwarding para a *ArrayQListWithSelection* que colocámos como atributo.

2. **Classe *AbsPlaylist*:**

   Nesta classe decidimos usar uma *ArrayList* para guardar as songs. Preferimos o uso da *ArrayList* invés do da *QListWithSelection*, devido à *ArrayList* possuir métodos que precisamos utilizar nos métodos *removeAtIndex* e *moveUpSelected*, pois a *QListWithSelection* não possui métodos para alterar ou remover elementos num certo índice, não possuindo então, uma boa implementação para os requisitos da nossa classe.

3. **Classe *PlaylistList*:**

   Nesta classe decidimos implementar a interface *QListWithSelection<Playlist>* e usar composição e forwarding com a classe *ArrayQListWithSelection*, em vez de estender a classe abstrata *AbsQListWithSelection<Playlist>*. Tomamos esta decisão para tornar mais flexível a troca da implementação de *QListWithSelection* usada pela *PlaylistList*, isto é, se quisessemos, numa iteração futura, alterar a implementação da *QListWithSelection* usada nesta classe, seria necessário redefinir métodos herdados pela *AbsQListWithSelecion*, ficando a implementação destes apenas dentro da classe *PlaylistList*, não podendo assim, usar noutras classes que também beneficiem desta nova implementação. Com isto, a classe fica menos dependente da herança, usando composição e forwarding para uma maior flexibilidade.

4. **Classe *Rate*:**

   Para esta classe utilizámos um enumerado em vez de uma classe, uma vez que, esta classe precisava que ser imutável e o enumerado já apresenta essa característica e também, já possui a interface *Comparable* implementada permitindo comparar os valores com ==.

5. **Método *Equals*:**

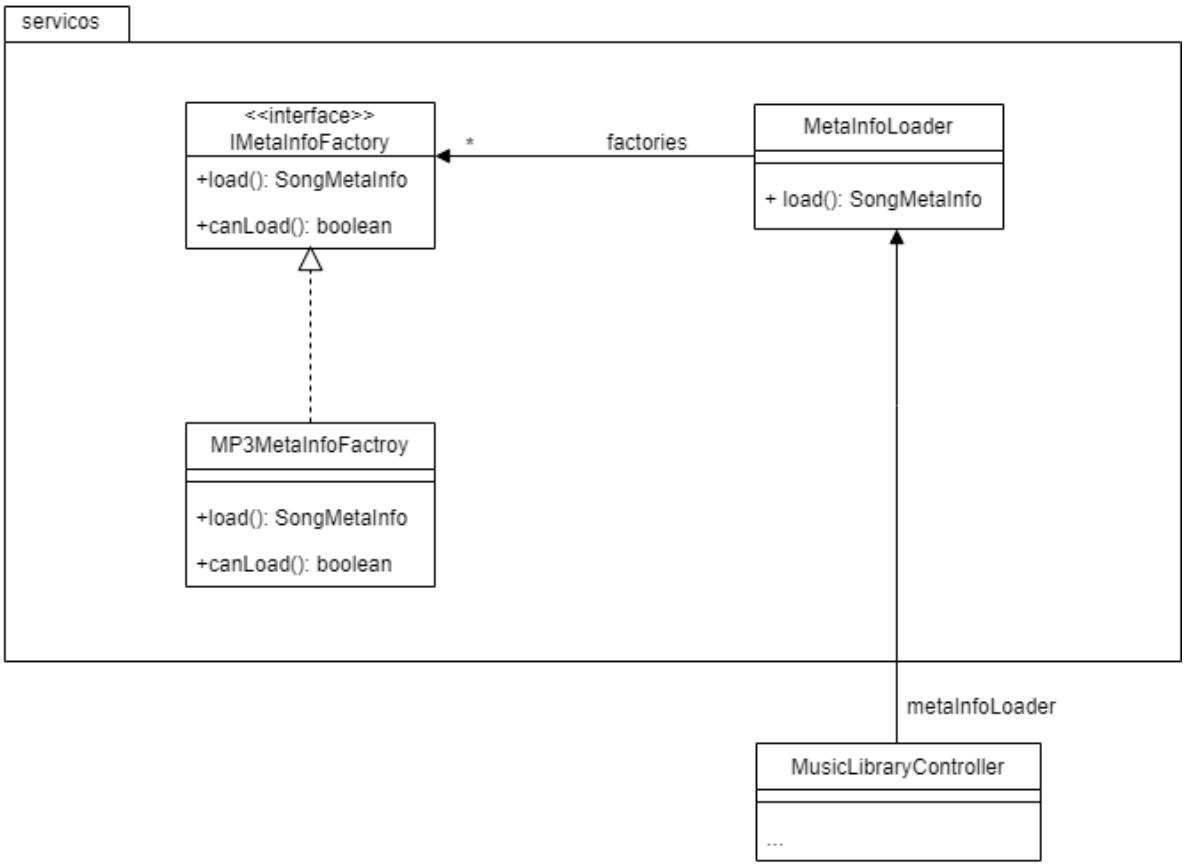
   Definimos equals apenas para a classe *Song*. Não foi necessário definir para *Rate* uma vez que, este é um enumerado, nem para a classe *SongMetaInfo*, uma vez que, sendo esta um record, já possui implementação deste método. Não implementámos o equals na classe *MusicLibrary* uma vez que, esta vai registar as suas playlists, logo apenas vai emitir eventos para as playlists que registou, de forma que as playlists não necessitam comparar se o evento emitido é da sua *MusicLibrary* ou não.

6. **Information Hiding:**

   No desenvolvimento do nosso programa utilizamos os princípios do *Information Hiding* para tornar os nossos métodos e atributos o menos acessível possível, não tornando públicos detalhes internos das implementações das classes. Usámos private em todos os atributos definidos (sendo estes apenas acedidos através de getters). No getter da *AbsPlaylist* que é usado na *MusicLibrary*, tornámo-lo *protected*, de modo a apenas estar acessível nas classes que estendam a *AbsPlaylist*, de forma a ser usado nas suas implementações internas.

7. **Package Serviços - Leitura da *SongMetaInfo*:**

   Para realizar a leitura da *SongMetaInfo*, utilizámos o padrão *Factory*, estando o desenho UML da nossa solução na imagem abaixo.



   Na nossa implementação temos:

   - Classe ***MetaInfoLoader:*** Classe que irá chamar o método *load* da factory que conseguir carregar os meta-dados a partir do path que for passado como argumento no método *load* desta classe.
   - Interface ***IMetaInfoFactory:*** Interface que representa factories usadas para criar *SongMetaInto*.
   - Classe ***MP3MetaInfoFactory:*** Que implementa a interface *IMetaInfoFactory* que irá dar load de meta-dados provenientes de um file MP3.

2 Observações:

1. **Java Modules:**

   Como extra sugerido pela professora Antónia na aula teórica, decidimos utilizar os módulos do Java no nosso projeto. Com isso colocamos nos exports os packages que seriam usados pelos clientes do nosso código, sendo estes o package do facade e os que possuem classes que sejam retornadas por estes.

2. **Testes:**

   Fizémos todos os testes que foram pedidos para as classes *ArrayQListWithSelection* e *Song*, e ainda alguns adicionais, todos eles com coverage de 100% no eclipse.

3. ***SimpleClient* output:**

   No nosso output do *SimpleClient*, nos meta-dados obtidos da song "O mundo é já aqui", observamos que apresenta o campo álbum preenchido, em vez de Unknown como no enunciado.

   Para além disso, colocámos uma seta (->) na song selecionada, que consta em sítios diferentes dos do output fornecido no enunciado, pois este último apresenta um typo.