Modelação e Padrões de Desenho

LEIRT 2024

Enunciado do Trabalho 2

Objectivos: Prática com aspetos funcionais da linguagem Java: expressões lambda, funções de ordem superior, construção de sequências *lazy* (`PipeIterable` e `Stream`).

Notas:

1. A solução entregue deve incluir todos os testes unitários necessários para validar o correto funcionamento das funcionalidades pedidas.
2. Este trabalho deve ser desenvolvido usando como base os módulos *gradle* music-all-streams e music-all-utils disponibilizados no repositório https://github.com/isel-leic-mpd/music-all-students-template.
3. Para utilizar a web API terá de obter uma chave em https://www.last.fm/api e criar um ficheiro de nome **lastfm_web_api_key.txt**

Fase 0

Antes de mais deverão fazer ***clone*** do repositório `music-all-students-template`, e depois copiar os módulos `music-all-utils` e `music-all-streams` para o vosso repositório, mantendo os nomes. Notem que terão de acrescentar esses módulos no ficheiro `settings.gradle` do vosso projeto. Deverão também remover do vosso projeto as pastas `.idea` e `.gradle` . Todas estas ações deverão ser realizadas não estando o vosso projeto aberto no `IntelliJ`.

Fase 1

1. music-all-utils - PipeIterable

Baseando-se nas implementações dadas nas aulas, implementar em `PipeIterable` as seguintes operações, preferencialmente pela ordem indicada, e os testes unitários que verifiquem o seu correto funcionamento:

a) Implemente o método *factory* **`iterate`** de modo a passar os respetivos testes unitários definidos em `PipeIterableTests`. Recomenda-se adicionar outros testes unitários.

b) Implemente o método terminal **`last`**, que retorna o último elemento da sequência a que for aplicado. Note que o tipo de retorno destes métodos é `Optional<T>`, para suportar sequências vazias.

c) Implemente a operação intermédia limit que produz uma nova sequência lazy com a dimensão máximo passda por parâmetro, e a operação **`skipWhile`**, que retorna uma nova sequência *lazy* que exclui os primeiros elementos da sequência **`this`** que verificam o predicado **`pred`**.

d) Implemente a operação intermédia **`cache`**, que retorna uma sequência lazy que representa os valores da sequência fonte. Independentemente do número de iterações que ocorram sobre a sequência cache, a fonte é iterada uma única vez. A solução terá de passar os testes unitários definidos em `PipeIterableTests`. Recomenda-se adicionar outros testes unitários. Em particular, note que para o teste unitário seguinte suceder é preciso garantir que as duas listas expected e atual são iguais, o que significa que a sequência inicial nrs so´pode ser consumida uma única vez.

```
Random r = new Random();
PipeIterable<Integer> nrs = generate(() -> r.nextInt(100));
nrs = nrs.cache();
var nrs1 = nrs.limit(10);
var expected = nrs.limit(10).toList();
var actual = nrs.limit(10).toList();
assertEquals(expected, actual);
```

Fase 2

2. music-all-streams - LastFmWebAPI

As classes que representam DTO e a classe **`LastFmWebAPI`** estão quase completamente implementadas, mas os métodos **`searchArtist`** e **`getAlbums`** de **`LastFmWebAPI`** deverão ser modificados para ignorar os artistas e álbuns que tenham identificador (**`mbid`**) nulo ou vazio, o que acontece frequentemente. Não deve usar ciclos nas alterações realizadas.

3. music-all-utils - Requests

Complete as classes **`CountRequest`** e **`SaverRequest`**, que são *decorators* de **`Request`** usados nos testes da **`LastFmWebAPI`**. O objetivo de **`CountRequest`** é contar o número de operações de **`get`** invocadas sobre o **`Request`** decorado e o de **`SaverRequest`** é guardar a resposta JSON devolvida sobre o **`HttpRequest`** decorado, de modo a que possa ser utilizada por uma instância de **`MockRequest`**. Na realização de **`SaverRequest`** utilize o método estático **`saveOn`** de **`MockRequest`**

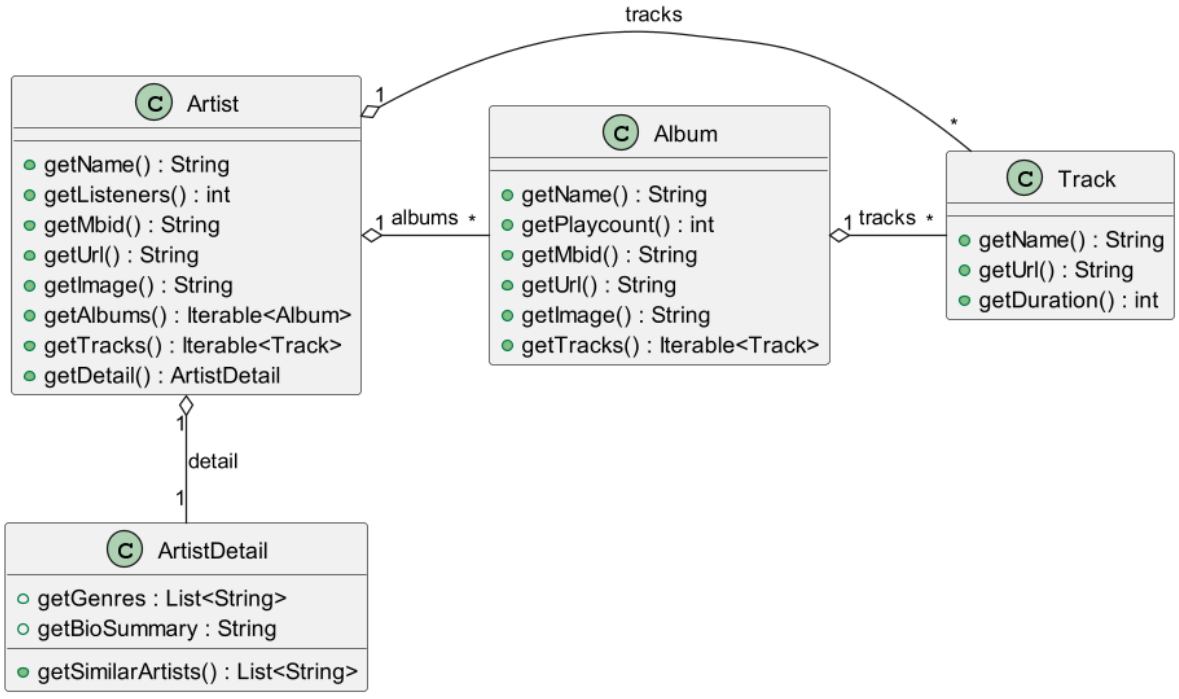
Fase 3

4. music-all-streams – Sequence

Implementar as operações **`skip`**, **`concat`** e **`zip`** em falta na interface **`Sequence<T>`** presente no *package* org.isel.music_all.streams.utils. Acrescentar à classe **`SequenceTests,`** dos testes do módulo, os testes que comprovem o seu correto funcionamento.

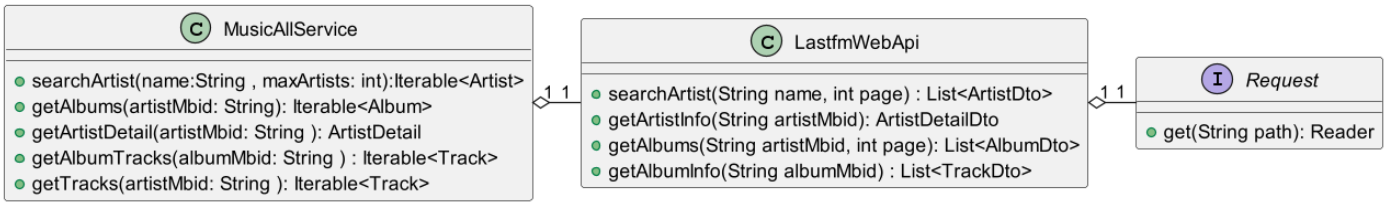
5. music-all-streams - MusicAllService

A classe **`MusicAllService`** implementa um serviço, sobre a last.fm API, que disponibiliza informação detalhada sobre artistas, álbuns e músicas. O modelo de domínio é formado pelas entidades: **`Artist`**, **`ArtistDetail`**, **`Album`** e **`Track`** e obedece à especificação apresentada no diagrama de classes seguinte:



As classes do modelo de domínio estão implementadas no *package* **model** de **music-all- streams** . Todas as relações entre as entidades de domínio são mantidas de forma *lazy*, no sentido em que só deverão espoletar pedidos à web API quando isso for estritamente necessário.

A instanciação e navegação dos objetos de domínio é feita por **`MusicAllService`** que recorre à classe **`LastFmWebAPI`** para realizar os pedidos à Web API de last.fm.



Complete os métodos referenciados de **MusicAllService** de modo a passar os testes unitários de **`MusicAllServiceTests`** (exceto os que estiverem comentados). Deve implementar outros testes unitários além dos fornecidos. Note que alguns dos métodos de **`LastFmWebAPI`** recebem um segundo parâmetro inteiro correspondente ao número da página a obter. Nestes casos o método correspondente de **MusicAllService** deve retornar uma *stream* que percorre os elementos de todas as páginas disponíveis até ser obtida uma página sem elementos ou ser atingido o limite de elementos especificado na chamada. Na implementação destes métodos não deve usar ciclos, mas apenas encadeamentos de operações com *streams*.

Fase 4

6. music-all-streams - StreamUtils

Na classe **`StreamUtils`** presente no *package* `org.isel.music_all.streams.utils`, implementar os métodos estáticos **`cache`**, que acrescenta às *streams* uma operação de funcionalidade semelhante à cache de `PipeIterable`, e **`intersection`**, que constrói uma *stream* a partir de duas outras *streams*, com os elementos que são comuns a ambas. Notem que o método **`intersection`** deve tirar partido do método **`cache`**. Deverão realizar testes para estes métodos, a colocar na classe **`StreamUtilsTests`** presente no *package* de testes.

Descomente o método de teste **`searchHiperAndCountAllResultsWithCache`** da classe **`MusicAllServiceTests`** que utiliza a operação de cache e verifique o seu correto funcionamento.

7. music-all-streams - MusicAllService

Tirando partido do método **`intersection`** desenvolvido na alínea anterior, completar acrescente ao serviço o método **`public Stream<String> commonArtists(String artist1, String artist2)`** que retorna uma *stream* com os nomes dos artistas que são comuns aos conjuntos de artistas similares a **`artist1`** e a **`artist2.`**

Datas limite de entregas:

Fase 2: 24 de abril. **Fase 4**: 8 de maio

Bom trabalho,
Jorge Martins