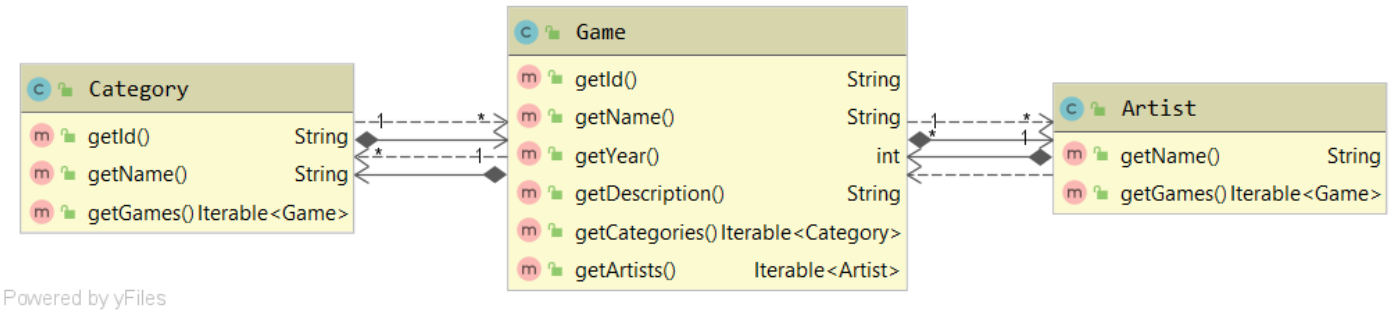
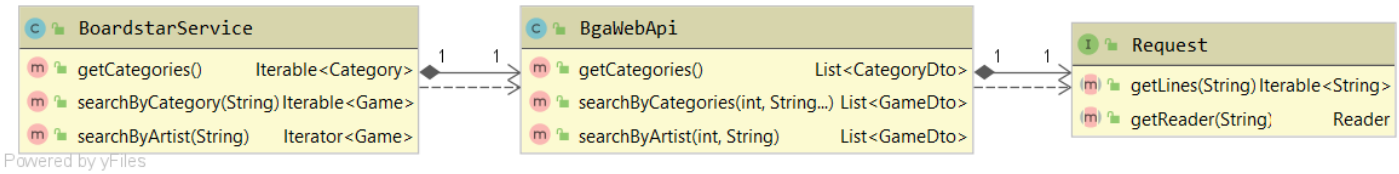
Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Modelação e Padrões de Desenho
2020

A biblioteca `boardstar-lazy` disponibiliza informação detalhada sobre jogos de tabuleiro, suas categorias e os seus artistas. Os dados são obtidos a partir de uma API RESTful: https://www.boardgameatlas.com/api/docs. O modelo de domínio é formado pelas entidades: `Category`, `Game` e `Artist` e obedece à especificação apresentada no diagrama de classes seguinte:



As classes do modelo de domínio estão implementadas no módulo **boardstar-lazy**. Todas as relações entre as entidades de domínio são mantidas de forma **lazy**.

A instanciação e navegação dos objectos de domínio é feita por `BoardstarService` que recorre à classe `BgaWebApi` para realizar os pedidos à Web Api de Board Game Atlas.



As funcionalidade de `BgaWebApi` recorrem às seguintes rotas da Web API de Board Game Atlas:

- https://www.boardgameatlas.com/api/docs/game/categories
- https://www.boardgameatlas.com/api/docs/search

Os resultados da API RESTFul podem ser convertidos através da biblioteca Gson para instâncias de classes pré-definidas (DTOs).

As relações entre as entidades do modelo de domínio (`Category`, `Game` e `Artist`) são mantidas de forma **lazy**. Por exemplo, a chamada a `getCategories()` não deve desencadear acesso de IO, tal como a chamada a `getGames()` sobre uma instância de `Category` também não deve fazer acesso IO até que os objectos `Iterator` sejam obtidos.

Note que alguns dos métodos de `BgaWebapi` recebem um segundo parâmetro inteiro correspondente ao número da página. Nestes casos o método correspondente de `BoardstarServive` retorna um iterável que percorre os elementos de todas as páginas disponíveis até ser obtida uma página sem elementos. Para tal deve executar um encadeamento de operações semelhante ao seguinte:

```
iterate(…) ==> 1,2,3,..
           ==> map(invoke BgaWebApi)
           ==> list 1, list 2, …
           ==> takeWhile(list.size() != 0)
```

```
==> flatMap(lst)
==> map(dto -> model)
```

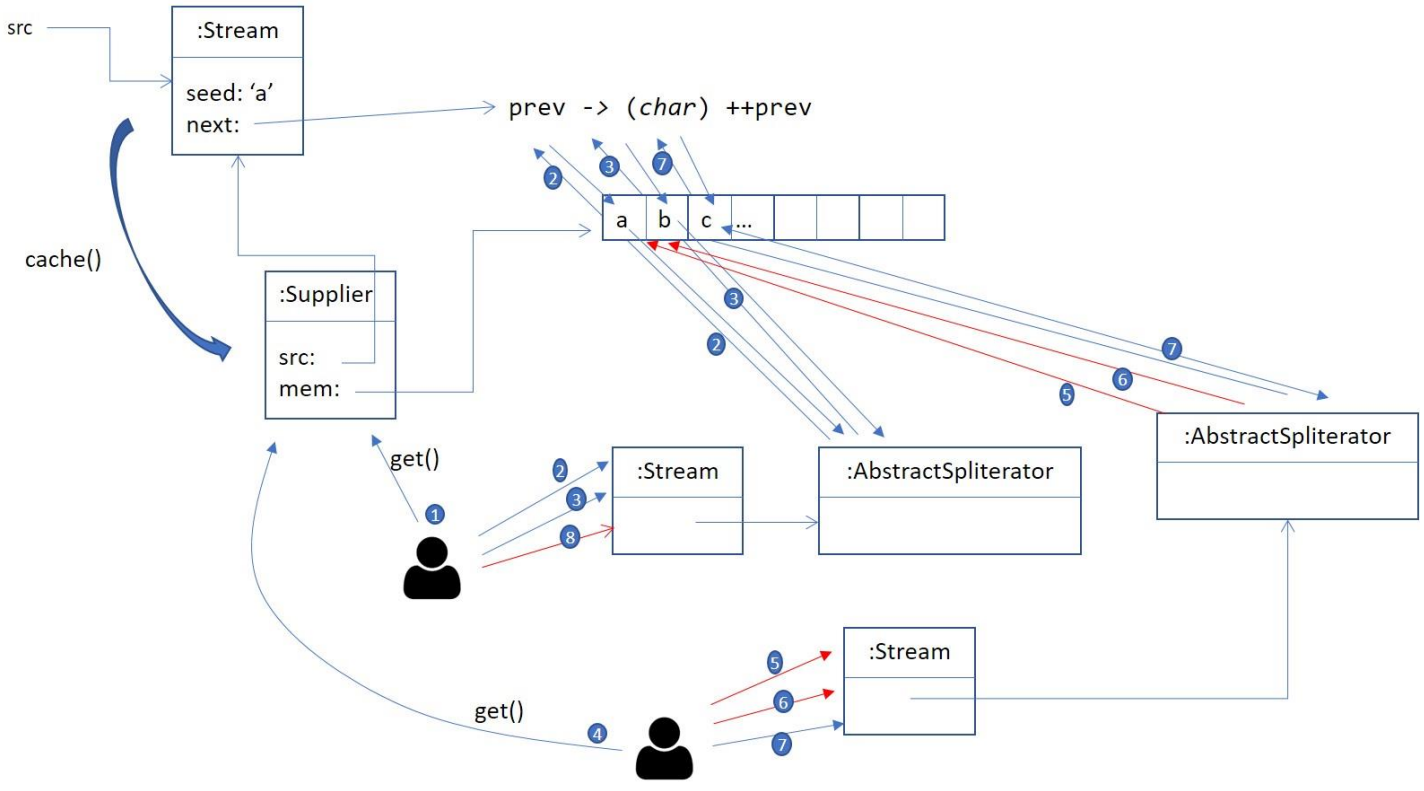
`LazyQueries` tem uma função utilitária `cache()` que pode ser aplicada a qualquer sequência retornando uma nova sequência do mesmo tipo, e.g. `games = cache(games)`. A sequência resultante deve guardar em memória os elementos que vão sendo obtidos por um iterador. O método `next()` retorna sempre os elementos que já estejam guardados em memória e só obtém um novo elemento caso este não esteja *cached*. Exemplo de utilização do método `cache()` sobre uma sequência infinita:

```
Random r = new Random();
Iterable<Integer> nrs = generate(() -> r.nextInt(100));
nrs = cache(nrs);
Object[] expected = toArray(limit(nrs, 10));
Object[] actual = toArray(limit(nrs, 10));
assertArrayEquals(expected, actual);
```

Note que os elementos das sequências resultantes de `cache()` podem ser obtidos alternadamente entre sequências, tal como apresenta o exemplo de intercalação de 8 acções na imagem seguinte:

- 1 Obtém uma sequênca do `Supplier` retornado por `cache()`.
- 2 e 3 Lê os valores `a` e `b` da sequência resultante de 1. que não existiam em memória e foram obtidos da sequência fonte (`src`) e adicionados em `mem`.
- 4 Obtém uma nova sequênca do `Supplier` retornado por `cache()`.
- 5 e 6 Lê os valores `a` e `b` da sequência resultante de 4. que já estavam em memória.
- 7 Lê o valor `c` da sequência resultante de 4. que não existia em memória e foi obtido da sequência fonte (`src`) e adicionados em `mem`.
- 8 Lê o valor `c` da sequência resultante de 1. que já estavam em memória.

```
src = Stream.iterate('a', prev -> (char) ++prev);
```



`StreamUtils` disponibiliza os métodos:

1. `Stream<T> interleave(Stream<T> src, Stream<T> other)` que retorna uma nova sequência com os elementos de `src` e `other` intercalados entre si. E.g. Dado `src = Stream.of("1", "2", "3")` e `other = Stream.of("a", "b", "c", "d", "e", "f")` então `res = interleave(src, other)` resultará numa sequência com os elementos `"1","a", "2, ""b", "3, ""c", "d", "e", "f"`. A implementação deve ser *lazy*.
2. `Stream<T> intersection(Stream<T> src, Stream<T> other)` que retorna uma nova sequência com os elementos iguais (segundo o `equals`) entre `src` e `other`. Faça um teste unitário que verifique o funcionamento desta operações fazendo a intercepção de duas sequências de jogos. e.g. `intersection(searchByCategory("War"), searchByCategory("Adventure"))`.

O módulo `boardstar-reactive` usa IO não-bloqueante, com uma interface `AsyncRequest` com um método `getBody()` que tenha API assíncrona. A implementação desta interface para pedidos HTTP GET recorre a uma biblioteca para realização de pedidos HTTP não bloqueantes, como o AsyncHttpClient.

Os métodos de `BgaWebApi` retornam resultados na forma de `CompletableFuture<List<...>>`.
Os tipos do modelo de domínio e serviço oferecem uma API assíncrona baseada no tipo Observable sempre que faça sentido.

Os métodos `searchBy...()` de `BoardstarService` e `getGames()` do modelo de domínio, recebem um parâmetro inteiro que determina o número total de elementos da sequência retornada, permitindo assim que possam ser desencadeados pedidos concorrentes à `BgaWebApi`. Ou seja, se for chamado o método `searchBy...(..., 270 )` de `BoardstarService` e se por sua vez `BgaWebApi` retornar resultados em grupos de 30 elementos, então serão feitos pedidos a `BgaWebApi` para as páginas de 1 a 9.

A aplicação Web usa a tecnologia com suporte para handlers assíncronos javalin.

A aplicação disponibiliza as seguintes páginas:

1. Listagem de todas as categorias de jogos. Cada categoria tem um link para a listagem de jogos dessa categoria (página 2).
2. Listagem de jogos de uma categoria ou de um artista. Cada jogo tem 2 links: um para a listagem dos seus artistas (página 3) e outro para a listagem das suas categorias (página 1 com query-string das categorias a apresentar). O número de items apresentados nesta página pode ser limitado por um parâmetro de *query-string*.
3. Listagem de artistas de um jogo. Cada artista tem um link para a listagem dos seus jogos (página 2).

As páginas anteriores são acessíveis através dos seguintes caminhos (paths):

1. `/categories`
2. `/categories/:id/games` ou `/artists/:id/games`
3. `/games/:id/artists`

A aplicação web **nunca poderá bloquear** (não fazer `join()` e nem `get()`) na obtenção de um resultado.
As listagens são retornadas no corpo da reposta HTTP em modo chunked (`response.setChunked(true)`) sendo a user-interface construída de forma progressiva à medida que a resposta é recebida no browser.

Por exemplo, o pedido a `/categories/eX8uuNlQkQ/games/` retorna cerca de 300 jogos que devem ser apresentados progressivamente à medida que são obtidos os resultados da Web API do Board Game Atlas.