Universidade do Minho
Escola de Engenharia

Relatório: Sistema de Gestão de Recomendações
UC: Laboratórios de Informática III
Grupo: 20

Constituição do grupo

★ Ana Rita Costa Teixeira - a93276
★ Beatriz da Silva Rodrigues - a93230
★ João Pedro Rodrigues Carvalho - a93166

Temas abordados:

★ Arquitetura de classes
★ Diagrama de classes
★ Testes de performance
★ Conclusões

Universidade do Minho
Escola de Engenharia

Índice

Arquitetura de Classes

A estrutura do programa está dividida de acordo com MVC, ou seja, em *model*, *view* e *controller*. A maior parte das classes estão acompanhadas de uma interface.

Model

★ Catálogo de utilizadores: classes User e Users

Para definir o catálogo de utilizadores, foram utilizadas duas classes: a classe *User* e a classe *Users*.

A classe *User*, que implementa a classe *UserInterface*, é constituída pelos construtores padrão, getters, setters, um método de validação e um método de parsing de utilizadores, quando lidos de um ficheiro de texto. É considerado que um utilizador é constituído pelo seu id, o seu nome e a sua lista de amigos. No entanto, na execução do programa, este último constituinte foi desprezado por indicação do enunciado.

A classe *Users*, que implementa a classe *UsersInterface*, contém uma hash table com todos os utilizadores. A sua key corresponde ao user id e o valor armazenado é uma *UserInterface*. É constituída por métodos que validam a existência de um dado utilizador ou o procuram e que devolvem quantos utilizadores existem armazenados. Para além disso, contém o método que realiza a leitura do ficheiro de utilizadores linha a linha, validando o user correspondente e, quando assim o for, o insere na hash table. Por fim, tem ainda um método que coleciona os user ids todos num set.

★ Catálogo de negócios: classes Business e Businesses

Para definir o catálogo de negócios, foram utilizadas duas classes: a classe *Business* e a classe *Businesses*.

A classe *Business*, que implementa a classe *BusinessInterface*, é constituída pelos construtores padrão, getters, setters, um método de validação e um método de parsing de negócios, quando lidos de um ficheiro de texto. Considera-se que um negócio é constituído pelo seu id, o seu nome, a sua cidade, o seu estado e as suas categorias.

A classe *Businesses*, que implementa a classe *BusinessesInterface*, contém uma hash table com todos os negócios. A sua key corresponde ao business id e o valor armazenado é uma *BusinessInterface*. É constituída por métodos que validam a existência de um dado negócio ou o procuram e que devolvem quantos negócios

foram armazenados. Para além disso, contém o método que realiza a leitura do ficheiro de negócios linha a linha, validando o negócio correspondente e, quando assim o for, o insere na hash table. Por fim, contém um getter para a hash table.

★ Catálogo de avaliações: classes Review e Reviews

Para definir o catálogo de avaliações, foram utilizadas duas classes: a classe *Review* e a classe *Reviews*.

A classe *Review*, que implementa a classe *ReviewInterface*, é constituída pelos construtores padrão, getters, setters, um método de validação e um método de parsing para avaliações, quando lidos de um ficheiro de texto. Considera-se que uma avaliação é constituída pelo seu id, pelo user id que a efetua, pelo business id do negócio à qual é destinado, o número de estrelas atribuídas, inteiros que caracterizam se é útil, engraçado ou fixe, a data de elaboração e o texto (caso o contenha).

A classe *Reviews*, que implementa a classe *ReviewsInterface*, contém uma hash table com todas as avaliações. A sua key corresponde ao review id e o valor armazenado é uma *ReviewInterface*. É constituída por métodos que validam a existência de uma dada avaliação ou a procuram e que devolvem o número de avaliações armazenadas. Para além disso, contém o método que realiza a leitura do ficheiro de avaliações linha a linha, validando a avaliação correspondente e, quando assim o for, a insere na hash table. As funções auxiliares às queries também estão contidas neste módulo. Acrescenta-se ainda que esta classe também é carregada com informação acerca dos negócios e dos utilizadores, decisão que será criticada nas conclusões do relatório.

★ Classe GestReviews

Na classe *GestReviews*, que implementa a *GestReviewsInterface*, para além de serem lidos os ficheiros de utilizadores, negócios e avaliações, são também implementadas todas as *queries*. Como variáveis contém um *UsersInterface*, um *BusinessesInterface* e um *ReviewsInterface*, ou seja, as estruturas responsáveis por armazenarem, respetivamente, os utilizadores, os negócios e as avaliações. Para além disso, também contém a função que permite guardar esta classe num ficheiro de objeto, enquanto a função que faz o carregamento se encontra na respetiva interface. Segue-se uma pequena explicação da lógica seguida na implementação das queries:

Query 1: Colecionam-se os negócios que já foram avaliados num set (para garantir que não há repetição) e depois cria-se uma lista que irá conter os negócios que não existem nesse set e, por isso, nunca foram avaliados.

Query 2: É utilizada uma classe auxiliar *Query2Pair* que permite armazenar dois inteiros, um que corresponde ao total de reviews e outro que corresponde ao total de utilizadores. No entanto, antes de realizar a query é necessário validar o mês. Para contar o número de reviews, apenas foi necessário uma variável que fosse incrementada por cada review relevante percorrida na hashtable de reviews, ao mesmo tempo que os users ids são adicionados a um set (para garantir que não há repetição). No final, o tamanho deste set corresponde ao número de utilizadores.

Query 3: Para armazenar a informação necessária para cada mês, através de um hashset, cada um foi ligado a uma estrutura definida pela classe *Query3Triple*. Primeiro, armazenam-se numa lista todas as avaliações que um utilizador efetuou. Depois estas são agrupadas por mês através de streams. Por fim, são calculados os valores relevantes para a query e armazenados na estrutura referida que é posteriormente ligada ao mês correspondente.

Query 4: De forma semelhante à query anterior, também se liga a cada mês uma classe auxiliar, desta vez *Query4Triple*. Primeiro, armazenam-se numa lista todas as avaliações relativas ao dado negócio. Depois estas são agrupadas por mês através de streams. Por fim, são calculados os valores relevantes para a query e armazenados na estrutura referida que é posteriormente ligada ao mês correspondente.

Query 5: É percorrida a hashtable de avaliações e, quando uma avaliação foi efetuado pelo utilizador dado, adiciona-se a uma hashtable o negócio e, caso este já exista, incrementa-se o número associado ou, caso não exista, adiciona-se o negócio e inicializa-se o inteiro a 1. É criada uma lista de pares a partir da hashtable que é ordenada por ordem decrescente do inteiro associado.

Query 6: É percorrida a hashtable de avaliações e, por cada avaliação, recolhe-se o seu ano e procura-se fazer a inserção numa hashtable que liga cada ano a outra hashtable que liga cada negócio ao número de avaliações recebidas. Caso não exista, a entrada é criada. Em simultâneo, os utilizadores que a efetuaram são organizados noutra hashtable semelhante, de forma a se saber para cada ano e para cada negócio, os utilizadores distintos que fizeram a avaliação. Assim, basta ordenar cada hashtable pelo número de avaliações e limitar o número de negócios a devolver por ano de acordo com o input recebido. Por fim, é adicionado o número de negócios distintos a cada negócio já ordenado por ordem decrescente de avaliações.

Query 7: É percorrida a hashtable de avaliações e, por cada avaliação, recolhe-se a sua cidade e adiciona-se o negócio para o qual remete, atualizando-se o número de avaliações numa hashtable. De seguida, por cidade, organiza-se a hashtable que liga os negócios ao número de avaliações, limitando que apenas se pretende os 3 primeiros; a lista resultante dos 3 negócios é colocada na hashtable final.

Query 8: É percorrida a hashtable de avaliações de forma a preencher uma hashtable que liga um user a um set com os negócios que avaliou. De seguida, são colocados numa lista de pares o nome do utilizador e o tamanho do set e esta é ordenada de acordo com o segundo valor e filtrada de acordo com o input.

Query 9: É percorrida a hashtable de avaliações de forma a preencher uma hashtable que liga um negócio a um set com os utilizadores que o avaliaram. De seguida, esta hashtable auxiliar é usada para calcular a média das classificações, guardada numa lista de pares Negócio-Média da avaliação. Esta lista é posteriormente ordenada e filtra-se apenas o número pretendido de acordo com o input.

Query 10: É percorrida a hashtable de avaliações de forma a preencher duas hashtables: uma armazena o número de avaliações por negócio e outra organiza os negócios por estado e cidade, associando-os também ao total de estrelas. No final, as duas são usadas para dividir o total de estrelas pelo número de reviews, cumprindo o objetivo.

Estatísticas 1: Coleciona a partir dos catálogos a informação pretendida no enunciado, armazenando-a num hashtable. Dentro do controller, é acrescentada também a informação acerca do filepath dos últimos ficheiros carregados e o tempo que este processo demorou.

Estatísticas 2: Foram definidas vários métodos auxiliares que calculam o pretendido para este caso nesta classe, como a *statsTotalRevByMonth()* e a *statsUsersRevByMonth*() por exemplo. No controlador, estes métodos são utilizados para transmitir à camada view os resultados obtidos.

★ Classe Clock

A classe *Clock* disponibiliza funções que permitem contabilizar tempos de execução.

★ Exceções

Foram criadas exceções para quando um utilizador não existe designada *UserDoesntExistException*, para quando um negócio não existe designada *BusinessDoesntExistException* e para quando um mês introduzido é inválido designada *EmtpyQueryException*.

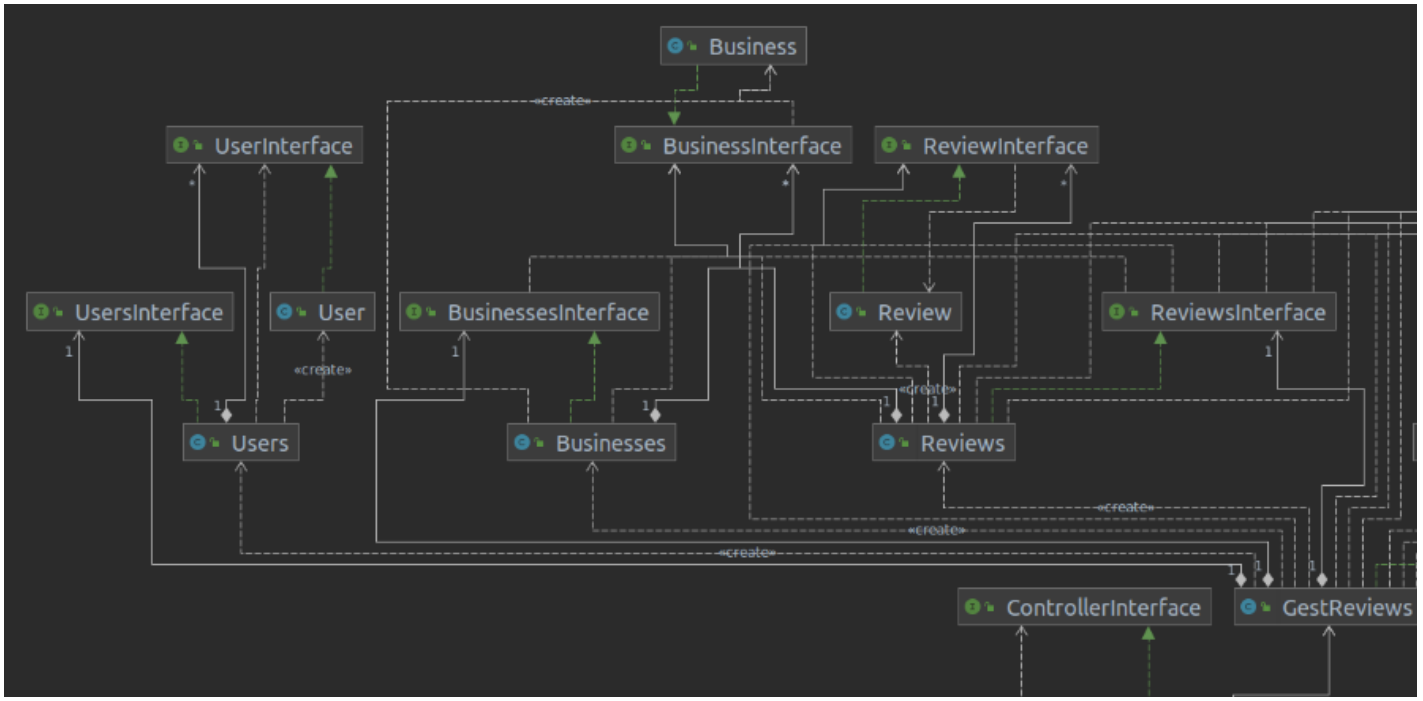
Controller

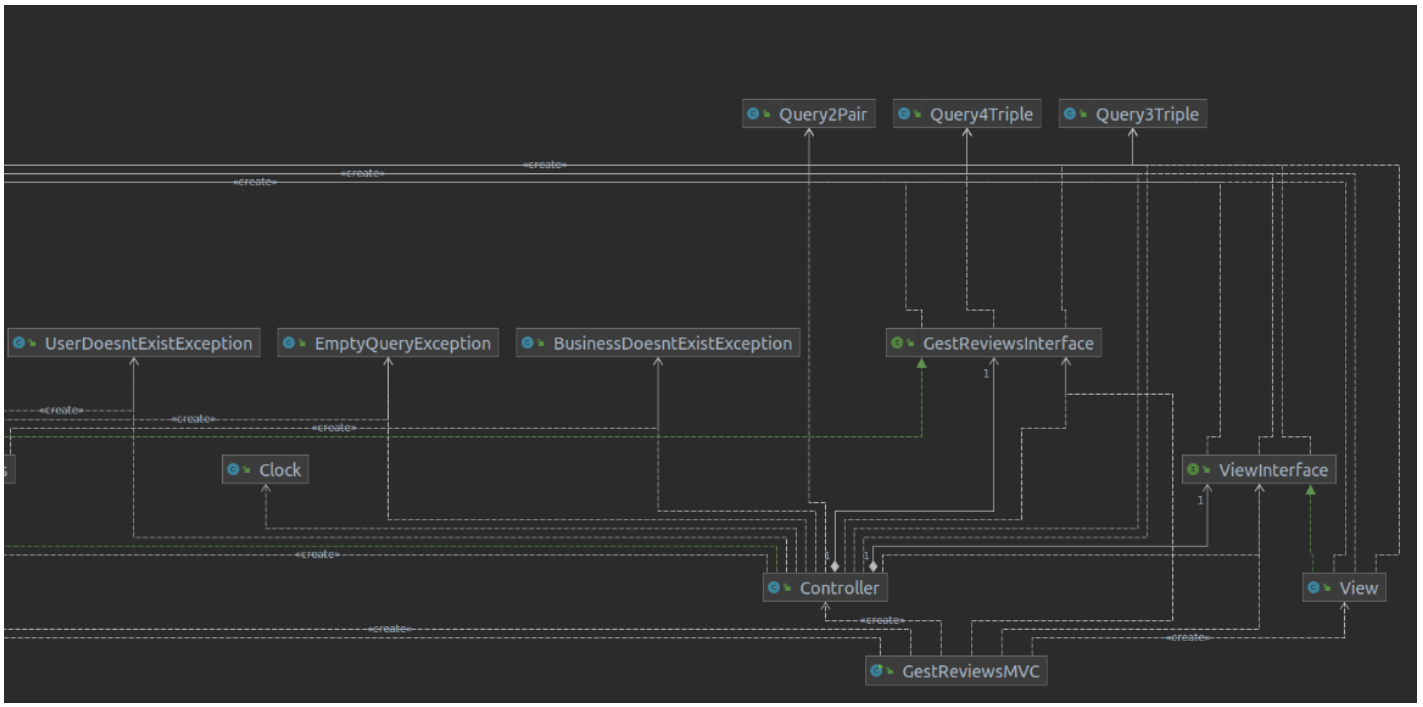
A classe *Controller*, que implementa a *ControllerInterface*, é responsável por controlar o funcionamento do programa. Permite o carregamento de ficheiros predefinidos ou customizados pelo utilizador. De seguida, permite ao utilizador escolher uma query, receber estatísticas, carregar o programa de ou guardá-lo para um ficheiro objeto. Os cálculos dos tempos de execução são realizados nesta classe também. As estatísticas previamente mencionadas também são tratadas parcialmente nesta classe.

View

A classe *View*, que implementa a *ViewInterface*, é constituída por métodos responsáveis por imprimir todo o output na tela do utilizador. Este output pode ser mensagens de erro, menus de guia de utilização, resultados a pedidos e tempos de execução. Permitem interagir com o utilizador de maneira mais sugestiva facilitando o uso do programa. Também é por esta camada que se recebe grande parte do input. Foi realizada paginação para a query 1 e para a query 5.

Diagrama de classes





Nota: A segunda imagem é a continuação horizontal da primeira.

Testes realizados

Apresentam-se agora os testes de performance realizados. A tabela seguinte contém os tempos de leitura, de cada query e de cada query estatística.

	Tempos (s)	Tempo médio (s)
Leitura de ficheiros (default)	8.56943646	8,777926418
	8.687113936	
	9.077228859	
Stats 1	0.859470602	0,809790471
	0.790819476	
	0.779081335	
Stats 2	2.031128452	2,071928639
	2.081316019	
	2.103341446	
Query 1	0.266424485	0,280032213
	0.284404773	
	0.289267381	
Query 2 (mês: 4, ano: 2020)	0.326338779	0,309101677
	0.331916895	
	0.269049356	
Query 3 (user-id: q_QQ5kBBwlCcbL1s4NVK3g)	0.099795522	0,102815331
	0.098761495	
	0.109888977	

Query 4 (business-id: 6iYb2HFDywm3zjuRg0shjw)	0.09012144	0,090032392
	0.088921336	
	0.091054401	
Query 5 (user-id: q_QQ5kBBwlCcbL1s4NVK3g)	0.098361102	0,093070136
	0.090160014	
	0.090689293	
Query 6 (X: 10)	1.137525168	1,08988101
	1.07470842	
	1.057409442	
Query 7	0.455130169	0,47282599
	0.437156175	
	0.526191625	
Query 8 (X: 10)	0.630113303	0,62317895
	0.612675661	
	0.626747886	
Query 9 (business-id: 6iYb2HFDywm3zjuRg0shjw, X: 10)	0.090684967	0,102312017
	0.091121399	
	0.125129684	
Query 10	0.659054729	0,616602313
	0.599552711	
	0.591199499	

A seguinte tabela revela a influência da mudança da estrutura de dados usada (*hash table*), em alguns tempos do programa.

	Hashtable	HashMap		TreeMap	
	Tempo médio (s)	Tempos(s)	Tempo médio (s)	Tempos(s)	Tempo médio (s)
Leitura de ficheiros (default)	8,777926418	9.326770037	8,859320754	12.287477867	12,235780132
		8.82567307		12.223469835	
		8.425519154		12.196392695	
Query 1	0,280032213	0.274955215	0,299554774	0.22474273	0,244740824
		0.344051548		0.251305431	
		0.279657558		0.258174311	
Query 4 (bus-id: 6iYb2HFDywm3zjuRg0shjw)	0,093070136	0.09179291	0,103396537	0.091528525	0,096373281
		0.095834555		0.095686307	
		0.122562147		0.101905012	
Query 7	0,47282599	0.485145739	0,484346831	0.990266527	0,952844902
		0.495890849		0.935514402	
		0.472003906		0.932753777	
Query 8 (X: 10)	0,62317895	0.652725699	0,657250468	0.709250706	0,696192561
		0.667179986		0.672867469	
		0.651845719		0.706459508	
Query 10	0,616602313	0.605959131	0,603086205	1.070024297	1,150567005
		0.589949667		1.174159275	
		0.613349817		1.207517443	

Dos testes acima foi concluído que a decisão de usar hashtables era a solução mais eficiente, tanto quanto à execução da leitura como quanto à execução de queries.

★ **Características do computador onde os testes foram corridos:**

```
Memória 15,5 GiB
Processador Intel® Core™ i7-10750H CPU @ 2.60GHz × 12
Capacidade de Disco 512,1 GB
Placa Gráfica NVIDIA GeForce GTX 1650
```

Conclusões

Em geral, os requisitos do enunciado foram cumpridos. No entanto, é importante destacar uma prática errada que foi aplicada: a classe reviews ter acesso ao catálogo de negócios e à lista de utilizadores de forma direta trata-se de um desrespeito ao encapsulamento e à estrutura do programa. A maneira correta de aceder-lhes seria com os respetivos getters dentro dos catálogos aos quais pertencem. Tratou-se de uma solução apressada por falta de tempo, havendo consciência de que não é de todo a maneira correta.

À parte dessa situação, não foi encontrado pelo grupo mais nenhum erro grave.