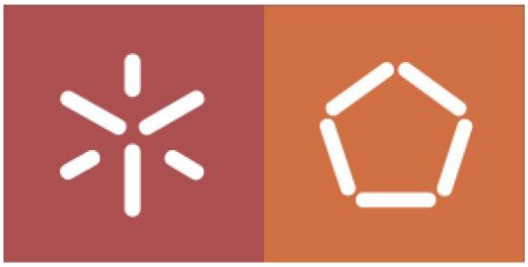

Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Projeto Java – GestReviews

Grupo 80

José Miguel Dias Pereira A89596

Joana Veiga de Oliveira A87956

Índice

Introdução

Este projeto, realizado no âmbito da Unidade Curricular de Laboratórios de Informática III, tinha como objetivo a criação de uma aplicação Desktop capaz de realizar consultas interativas de informações relativas à gestão básica de um sistema de recomendação/classificação de negócios. À semelhança do projeto realizado anteriormente em C, também aqui nos deparamos com a programação em grande escala, grandes volumes de dados e conceitos como o encapsulamento. Além dos conceitos mencionados anteriormente há ainda a acrescentar o da programação com interfaces.

Arquitetura de aplicação

O projeto segue a arquitetura de MVC(Model-View-Controller). Além disso, temos ainda a acrescentar uma aplicação independentemente que nos permite realizar testes de perfomance ao nosso programa.

Model

User

```
private String id;
private String name;
```

Classe que armazena informação relativa a um utilizador, tal como o seu id e nome. Esta contém métodos que permitem obter o nome e o id de um dado utilizador, bem como o método *userFromLine que* permite criar um user, a partir de uma linha de um ficheiro com informações de users.

Users

```
private Map<String, IUser> users;
```

Catálogo que guarda todos os utilizadores com o auxílio de um Map. A sua key é o id do user que se encontra associado como value. Aqui estão definidos métodos para adicionar utilizadores, ver se um dado utilizador está no catálogo, bem como para, a partir de um ficheiro, ler as informações relativas aos users e preencher o catálogo.

Business

```
private String id;
private String name;
private String city;
private String state;
private List<String> categories;
```

Classe que contém as informações de um negócio como o seu id, nome, cidade, estado e as categorias a que pertence. Aqui podemos encontrar métodos que permitem obter qualquer uma destas informações, tal como o método *businessFromLine* que é semelhante ao já referido na classe *User.*

Businesses

```
private Map<String, IBusiness> businesses;
```

Catálogo que, através de um Map, associa cada id ao negócio correspondente. Os seus métodos permitem, além de adicionar novos negócios e verificar a existência de um negócio, também preencher o catálogo com as informações contidas num ficheiro.

Review

```
private static final String dateFormat = "yyyy-MM-dd HH:mm:ss";//2014-10-11 03:34:02
private static final SimpleDateFormat dateParser = new SimpleDateFormat(dateFormat);
private String id;
private String userID;
private String businessID;
private float stars;
private int useful;
private int funny;
private int cool;
private LocalDate date;
private String text;
```

Classe que guarda as informações relativas a uma review tal como o seu id, o id do utilizador que a realizou, o id do negócio que foi avaliado, o número de estrelas atríbuidas, o quão *useful, funny* ou *cool* era o negócio, a data em que foi realizada, bem como algum comentário extra. Tal como as classes *User* e *Business* contém métodos para obter as respetivas informações e o método *reviewFromLine* que permite obter uma review através de uma linha de ficheiro.

Reviews

```
private Set<IReview> reviews;
```

Catálogo que contém todos as reviews armazenadas num Set. Aqui é possível adicionar novas reviews, preencher o catálogo a partir de um ficheiro, calcular o número de reviews por utilizador, bem como obter um Map que associa a cada mês todas as reviews realizadas nesse mesmo mês.

GestReviews

```
private IUsers users;
private IBusinesses businesses;
private IReviews reviews;
private List<String> ultimosFicheirosLidos;
private int reviewsErradas;
private Query10 query10;
```

Classe principal que agrega as mencionadas acima e contém ainda uma lista com os últimos ficheiros lidos, o número de reviews erradas e as informações relativas à query10.

Esta classe contém os métodos relativos a cada query, bem como os métodos que permitem obter as estatísticas gerais e de ficheiros.

View

Camada que apresenta ao utilizador os vários menus, informações sobre a interação com o programa, possíveis mensagens de erro, bem como permite, de uma forma mais organizada, apresentar grandes quantidades de informação provenientes das queries.

Controller

Camada responsável pela comunicação com o utilizador. Os pedidos do utilizador são aqui recebidos, enviados para o *Model* e após a sua conclusão é invocada a *View* para que o respetivo resultado seja apresentado.

Extra

Query2/ Query3Triple/ Query4/ Query5Pair/ Query6/ Query8/ Query9Pair/ Query10/ Rating Business

Classes extra que permitem facilitar e organizar os resultados das queries, quando necessário.

WriteReadGestReviews

Classe responsável por ler e escrever em ficheiro qualquer estado da aplicação.

Queries

Query 1

É percorrido o catálogo de reviews e é adicionado a um Set auxiliar o ID do negócio contido em cada uma. De seguida é percorrido o catálogo de businesses e, caso esse ID não esteja presente no Set anterior este é adicionado a um novo TreeSet, que se encontra ordenado por ordem alfabética, e irá ser depois retornado.

Query 2 (número total de reviews num dado mês e ano)

A lógica da query 2 pode ser dividida nos seguintes passos

- Criar um **HashSet** que conterá os Users ids de todos os reviewers (sendo set não irá ter elementos repetidos)
- Percorrer os reviews todos
- Para cada review verificar se sua data tem mês e ano igual ao fornecido
- Com o numero total de reviews e o set preenchido criar uma instância da classe Query 2.

A classe query 2 tem como propriedades o que é pedido para a query.

- totalReviews – total número de reviews
- totalUsersUnique – total de users únicos reviewers desse mês e ano

Query 3

Primeiramente é invocado o método contido na classe *Reviews* que retorna um Map organizando as reviews mês a mês. Posto isto, percorremos esse Map e em cada mês, percorremos o set de reviews associado. Caso o ID do utilizador esteja contido em algumas delas, aumentamos o número de reviews, adicionamos o número de estrelas da respetiva review e aumentamos o total. Antes de passarmos para o próximo mês , calculamos a classificação média dividindo as estrelas pelo total e recorremos à classe extra chamada *Query3Triple*, criando uma instância da mesma e adicionando-a ao Set que irá depois ser retornado.

Query 4 (dado um business ID, determinar mês a mês as suas avaliações)

Esta query pedia várias métricas, pelo que se teve de se estruturar o resultado de forma a que a amostragem do resultado fosse eficiente. Ora a query pede para termos a informação de cada business organizada mês a mês, e para cada mês quantas vezes foi avaliado (por users únicos) e por fim a média da classificação.

Começamos por descrever a estrutura de dados escolhida para o armazenamento, de seguida como o seu preenchimento é feito, e por final como é apresentado ao user o resultado

Estrutura

Para garantir a informação organizada mês a mês guarda-se o resultado da query num TreeMap, onde a chave é o mês o valor é a classe auxiliar RatingBusiness que permite guardar diversa informação para cada business.

Preenchimento da estrutura

- Instanciar query 4 (que inicializa um Treemap como acima referido)
- Percorrer os reviews todos
- Para cada review verificar se o seu BusinessId é igual ao fornecido
    - Se sim:
        - Obtemos o mês, o nº de estrelas, e o userId (reviewer)
        - Verificar se o TreeMap contem ou não a chave de valor ***mês***
            - Se sim
                - Obtemos o ratingBusiness respetivo e faz-se as seguintes operações
                    - Adicionar o UserID ao seu HashSet (únicos)
                    - Incrementar o contador de reviews
                    - Adicionar a atual avaliação (nº estrelas)
            - Se não
                - Cria-se um novo RatingBusiness com
                    - Com o userId atual
                    - Contador de reviews a 1
                    - E com a atual avaliação (nº strars)
    - Se não
        - Não fazer nada e avançar para o próximo review

Impressão/amostragem do resultado

Com a estrutura acima descrita já preenchida, a impressão basicamente é percorrer os pares chave-valor do TreeMap e para cada chave (Mês) imprimir do respetivo valor (RatingBusiness) o número de reviews, total de users únicos (count do HashSet dos users), e a média de avaliação que a partir da Lista de avaliações no RatingBusiness calcula a média.

Query 5

Percorremos as reviews e, caso o userID de uma dada review, seja igual ao passado como parâmetro, obtemos o nome do negócio associado ao businessID dessa mesma review. Este é depois adicionado a um Map auxiliar que associa a cada negócio o número de vezes que foi avaliado pelo utilizador em questão. Cada par é posteriormente transformado numa instância da classe extra Query5Pair e adicionado ao Set de retorno.

Query 6

Para esta query temos a classe Query6.

O processamento desta query é dividido em duas partes, primeiro o preenchimento do TreeMap **TreeMap<Integer, TreeMap<String, RatingBusiness>>,** onde no primeiro Treemap a chave é ano, e o valor é outro TreeMap onde a chave é o BusinessId e o valor um RatingBusiness:

- Instanciar a classe Query6
- Percorrer todos os reviews
- Para cada review obter o BusinessId, ano e UserId
- Começar por verificar se o ano existe na estrutura acima referida

No final, a partir do TreeMap acima preenchido criamos um outro TreeMap

TreeMap<Integer, TreeSet<RatingBusiness>> onde a chave é o ano, e o valor é um treeSet de RatingBuinessses ordenado pelo seu número de reviews (criado um comparator para o propósito **RatingBusinessByTotalReviews**

Nota: O primeiro TreeMap é preciso porque se usássemos desde início o segundo (com um TreeSet como valor) a cada review processado, ter-se-ia que para o ano do review a ser iterado, iterar o TreeSet à procura do RatingBusiness que tivesse como propriedade o BusinessId atual, o que causou imensos problemas de "performance".

Query 7

É percorrido o catálogo das reviews, e através do businessID associado a cada uma obtemos o negócio correspondente. Posto isto, temos um Map auxiliar onde a cada cidade associamos outro Map que associa a cada negócio o número de reviews. O negócio é assim associado à cidade correspondente, aumentando também o número de reviews. Para finalizar, todas as cidades são novamente percorridas e, para cada uma, é criada uma lista de nomes de negócios, ordenada decrescentemente pelo número de reviews de cada um e com tamanho igual a 3. A cidade e a lista correspondente são assim adicionadas a um Map que irá ser depois retornado.

Query 8

Como esta classe pede o top de Users que avaliaram mais negócios **diferentes**, a escolha de estrutura foi em função disto. Usamos duas estruturas diferentes, uma para durante a iteração dos reviews, e outra para facilitar a amostragem.

Primeiro preenchemos um HashMap onde a chave é o UserId e o valor é um HashSet de strings que corresponde aos businesses Ids

Map<String, HashSet<String>> Ora no fim da iteração dos reviews todos, temos a informação guardada para User uma coleção de todos os seus Businessses classificados (**únicos**)

Depois deste preenchido fazemos a conversão desta para um **TreeMap<Integer,String>** onde a chave é o número total de businesses diferentes classificados, e o valor é o UserId do utilizador respetivo

Ter a informação neste último TreeMap facilita imenso na altura da impressão do resultado.

Query 9

As reviews são percorridas e, caso o businessID de alguma seja igual ao passado como parâmetro, obtemos o utilizador associado a essa mesma review. Este é depois adicionado a um Map auxiliar que associa a cada user uma lista com as várias classificações atribuídas a esse

negócio. Posto isso, este Map é percorrido e é calculada a classificação média para cada utilizador com base na sua lista, ao mesmo tempo que é criada uma instância da classe extra Query9Pair, que contém um utilizador e a respetiva classificação. O seu retorno é um Set, ordenado decrescentemente conforme a classificação e com tamanho igual ao especificado como argumento.

Query 10

Antes de mais, como esta query não depende de nenhum parâmetro, isto é, seja ela chamada uma ou mais vezes o resultado é sempre o mesmo, decidimos defini-la como propriedade da classe GestReviews. Desta forma, se for a primeira vez chamada então todo o processamento é feito, mas se já tiver sido chamada o seu resultado anterior é aproveitado, poupando assim tempo.

Estrutura

Map<String, Map<String, Map<String,RatingBusiness>>>

- **State1**
  - **CityA**
    - **Business43**
      - **RatingBusiness**
    - **Business89**
  - **CityB**
    - **Business 12**
      - **…**
    - **Business19**
      - **..**
  - **…**
- **State2**
  - **….**
  - **…**

Embora o map acima parece demasiado complexo, foi a melhor maneira de conseguirmos de forma eficaz, review a review organizar a informação de modo requisitado. Os maps são extremamente eficazes pelo que foram altamente usados ao longo do projeto.

Lógica

- Começamos por verificar se a propriedade query10 é ou não nula. Se sim, percorremos os reviews e para cada um obtemos as stars e o business. De seguida verificamos se o city exite.
- Se existir verificamos se o state já exite, em caso positivo, verificamos se business existe, se sim adicionamos métrica ao RatingBusiness, se não criamos um novo RatingBusiness.
- Se o city não existir adiciona-se uma nova entrada como city e criar rating business.
- Se o state não existir adicionar nova entrada como novo state, novo city e novo ratingBusiness.
- Se a query 10 não for null retorna-se o valor da propriedade.

Escrita e Leitura de dados em ficheiro de objetos

Para a escrita e leitura de dados em ficheiros de objetos criou-se uma classe auxiliar (**WriteReadGestReviews**) com dois métodos, um para a escrita (**WriteGestReview**) e outro para a leitura (**ReadGestReview).**

No caso da escrita:

1. Cria-se um **FileOutputStream** passando-lhe a diretoria do ficheiro destino
2. Cria-se um **ObjectOutputStream** passando-lhe o **FileOutputStream** criado acima
3. Invocar o método **WriteObject** do **ObjectOutputStream** passando como parâmetro a instância do **GestReviews**

No caso da leitura:

1. Cria-se um FileInputStream passando-lhe a diretoria do ficheiro a ler
2. Cria-se um BufferedInputStream para a leitura ser feita em blocos
3. Cria-se um ObjectInputStream passando-lhe o BufferedInputStream acima criado

Testes

Toda a lógica feita no GestReviews é testada. Para cada teste usou-se a classe disponibilizada Crono. Nesta secção vamos mostrar como foram feitos os testes para a leitura dos ficheiros CSV, escrita e leitura em ficheiro de objetos e computação das queries.

A classe Tests tem duas propriedades.

- IGestReviews - necessária para invocarmos os métodos a testar. O valor é inicializado a NULL, e é preenchido apenas a primeira vez que for preciso, seja em que teste for.
- IViews – necessária para gerir a impressão dos menus/opções

Leitura dos ficheiros CSV

FICHEIRO	TEMPO (EM SEGUNDOS)
3 FICHEIROS EM SIMULTÂNEO	26.6
USERS.CSV	16.7
BUSINESSES.CSV	0.42
REVIEWS.CSV	9.77

Para cada teste é chamada o método de leitura da sua classe respetiva. No caso do teste isolado da leitura do reviews.csv, como o preenchimento da classe *Reviews* depende de já ter os *Users* e os *Businesses* carregados para validações acima referidas, faz-se o seu carregamento primeiro, mas apenas é contador apenas é inicializado quando irá ser feito o carregamento das *Reviews*.

Escrita e leitura em ficheiros de objetos

No caso do teste de escrita é preciso que os dados tenham sido previamente carregados em memória, se tal ainda não foi feito é então feito antes do teste, e o cronómetro só é começado efetivamente quando a escrita é feita.

No caso do teste de leitura o ficheiro tem de existir.

LEITURA/ESCRITA	TEMPO (EM SEGUNDOS)
LEITURA	16.2591941
ESCRITA	97.4637305

Queries

Para cada teste de uma query são feitas 2 verificações:

- Se os dados já estão carregados em memória
  - Se sim, é feito o carregamento (sem contar para as métricas do teste)
  - Se não, é inicializado o teste
- Se o utilizador pretende usar parâmetros específicos, caso a query assim o exija.
  - Se sim, então é lido do STDIN o input
  - Se não, são usados os valores default definidos como propriedade da classe Tests

QUERY	TEMPO (EM SEGUNDOS)
1	2.510379
2	0.7488384
3	0.6590971
4	0.8265151
5	0.8957809
6	3.6232593
7	2.2051919
8	1.8939988
9	1.4486866
10	1.8056998

Parâmetros utilizados:

Query 2

Mês: 11, Ano: 2020

Query 3

User ID: RtGqdDBvvBCjcu5dUqwfzA

Query 4

Business ID: dr9PgB_kTp998OYWixJYAw

Query 5

User ID: RtGqdDBvvBCjcu5dUqwfzA

Query 6

Top: 10

Query 9

Business ID: dr9PgB_kTp998OYWixJYAw

Conclusão

Em termos gerais estamos bastante satisfeitos com o resultado final do projeto. Todos os requisitos foram cumpridos, e conseguimos ainda realizar a parte complementar de testes que ficou completamente funcional, tal como foi apresentado acima.

O desenvolvimento do projeto correu bem, sem grandes percalços. Java é uma linguagem bastante mais fácil de programar do que C, uma vez que nos abstrai bastante de alocações de memória.

Diagrama de Classes