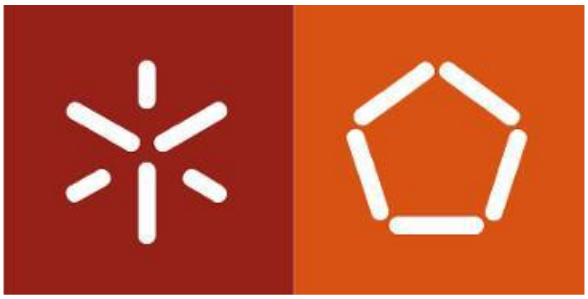

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática 3

Projeto: Sistema de gestão e consulta de recomendações de negócios na plataforma Yelp

Grupo 75

Arthur Cardoso Franca (a80011)

Duarte Nuno Pereira Moreira (a93321)

Lucas da Silva Carvalho (a93176)

Ano 2020/21

Índice

Introdução e desafios

Este projeto foi realizado no âmbito da disciplina de LI3 e, à semelhança do projeto anterior, consiste na gestão de um grande volume dados tornando-se assim num projeto de programação em larga escala. A linguagem de programação utilizada é o Java.

Afastando-se de longe dos problemas encontrados no primeiro projeto tais como, problemas de memória e como guardar a informação lida, este, o de java, não mostrou grandes dificuldades a esse ponto. Aqui, cremos que o principal obstáculo tenha sido obter uma correta organização da estrutura global do projeto, ou seja, sendo este realizado em java, surgiram questões como: que classes criar? será que esta deve herdar aquela? que variáveis de instância utilizamos? Depois de ultrapassar este problema (que acreditamos não ter sido resolvido com a maior eficácia), toda a construção do trabalho fluiu sem grandes obstáculos. Para além disso, e mais uma vez, o fator “boa gestão de tempo” mostrou ser fatal neste projeto.

Classes

a. GestReviewsAppMVC

Esta é a classe principal e apenas tem a função de arrancar o programa. Para tal o método main invoca o método run da classe Controller.

```
public class GestReviewsAppMVC {
    public static void main(String[] args) {
        Controller.run();
    }
}
```

b. Controller

A classe Controller, como o próprio nome indica controla o fluxo do programa e gere os comandos dados pelo utilizador interagindo com as outras classes.

```
while (!exit) {
    int option;
    if (sgr.getFiles().isEmpty()) option = 13;
    else option = UI.Menu();
    switch (option) {
        case  0 -> exit = true; // leave
        case  1 -> {...} // stats 1
        case  2 -> {...} // stats 2
        case  3 -> {...} // query 1
        case  4 -> {...} // query 2
        case  5 -> {...} // query 3
        case  6 -> {...} // query 4
        case  7 -> {...} // query 5
        case  8 -> {...} // query 6
        case  9 -> {...} // query 7
        case 10 -> {...} // query 8
        case 11 -> {...} // query 9
        case 12 -> {...} // query 10
        case 13 -> {...} // load files
        case 14 -> {...} // save struct
    }
}
```

c. GestReviews

A classe GestReviews é responsável por guardar o estado interno do programa. Para tal, esta guarda informação do nome dos ficheiros lidos, assim como toda a informação lida dos ficheiros. Para além disso existe uma variável com o objetivo de contabilizar as reviews inválidas, aquando da leitura dos ficheiros.

```
// variaveis de instancia
private List<String> files;
private CatalogoU CU;
private CatalogoB CB;
private CatalogoR CR;
private int reviewsErradas;
```

1. Estatisticas

A classe Estatisticas é implementada de forma a herdar a classe GestReviews para que esta tenha acesso aos dados lidos dos ficheiros. Como variáveis de instância para esta classe foram criadas duas, uma que irá conter os resultados pedidos no ponto 1 das estatísticas (dados referentes aos últimos ficheiros lidos) e a outro com os do ponto 2 (dados registados nas estruturas).

```
// variáveis de instância
Map<String,Integer> sats1;
Map<Integer, List<String>> reviewsMes;
```

2.Queries

À semelhança da classe anterior a classe Queries também herda os dados da classe GestReviews. Para esta classe foi decidido não criar variáveis de instância, uma vez que não encontramos necessidade para tal. Aqui são criados dois métodos por cada query apresentada no enunciado. Uma que calcula os resultados pedidos e outra que faz o tratamento dos resultados (toString).

d/e/f. Catalogo(U/B/R)

Estas três classes são criadas de forma idêntica, variando apenas no objeto a ser tratado, podendo este ser um User, um Business ou uma Review. Cada uma destas classes guarda a informação lida de cada um dos ficheiros respetivos agrupando todas estas entidades em Maps em que a chave será o id lido e o valor será o(a) User/Business/Review respetivo(a). Como é de esperar existe aqui uma estratégia de composição com as classes User, Business e Review.

```
// variaveis de instância
private Map<String,User> users;
private int invalidos; // users mal formatados
```

g. User

A classe User tem como objetivo representar entidades do tipo user. Estes contêm um id e um nome. A classe para além dos métodos usuais apresenta um método de inicialização de um User a partir de uma String e outro de validação.

```
// variáveis de instância
private String id;
private String name;
```

h. Business

De forma idêntica a classe Business representa entidades do tipo business. Estes contêm um id, um nome, uma cidade, um estado e uma lista de categorias. Apresenta também, para além dos métodos usuais, um de inicialização a partir de uma String e um de validação.

```
// variáveis de instância
private String id, name, city, state;
private List<String> categories;
```

i. Review

Mais uma vez a classe Review será semelhante às duas anteriores diferenciando apenas nas variáveis de instância, esta apresenta, um id da review, um id do user que realizou a review, id do business alvo da review, e outros como stars, useful, funny, cool, date e text.

```
// variáveis de instância
private String id, user_id, business_id;
private float stars;
private int useful, funny, cool;
private String date;
private String text;
```

j. Leitura

A classe Leitura é uma classe muito simples e auxiliar que tem apenas como objetivo ler e armazenar as linhas lidas de um ficheiro escrito em modo texto.

```
// variaveis de instancia
private List<String> linhas;
```

k. UI

A classe UI, tem como função tratar da representação visual de menus, mensagens de erro/sucesso, entre outros. Resumindo, tem como objetivo mostrar o conteúdo relevante ao utilizador de forma que este tome decisões e escolha opções, que serão posteriormente recebidas pelo Controller.

Arquitetura

Este projeto foi desenvolvido segundo a estrutura Model View Controller (MVC), e por isso, encontra-se dividido em três principais camadas, independentes umas das outras:

- Camada de dados e algoritmos, designada MODEL;
- Camada visual de interação com o utilizador, como por exemplo, menus, listas de resultados, etc, designado VIEW;
- Camada responsável pela ordem de execução correta da aplicação em função dos desejos do utilizador, designada CONTROLLER;

Model: constituído pelas classes GestReviews, Estatisticas, Queries, CatalogoU, CatalogoB, CatalogoR, User, Business, Review e Leitura

View: constituído apenas pela classe UI.

Controller: constituído pelas classes GestReviewsAppMVC e Controller.

Toda a construção do projeto foi pensada de forma a respeitar as normas de programação orientada aos objetos, nomeadamente o encapsulamento e a abstração de implementação.

Complexidade das estruturas

Como visto anteriormente ao longo do projeto são usadas estruturas como Maps e Lists, no entanto, em algumas das queries eram pretendidos resultados ordenados o que nos levou à utilização de TreeSets que, claro, implicou a criação de Comparators. A estratégia utilizada ao longo das queries foi sempre muito semelhante e consistiu em guardar e devolver os resultados em Maps. No entanto, isto mostrou ser uma estratégia longe do ideal, uma vez que, em muitos casos recorreu-se a um encandeamento de Maps e até mesmo TreeSets, o que complicava ainda mais quando chegava a parte de ordenar os resultados contidos nestes Maps encadeados. Todos estes pontos contribuíram para que o código se tornasse de difícil interpretação, contudo e apesar disso os resultados obtidos foram os esperados.

Ilustração de alguns exemplos dos problemas referidos:

```
Map<Integer, Map<String, Map<String, Integer>>> resultsIds = new HashMap<>();
//  Map< ano    , Map<businessId, Map<userId, 1>>
```

Encandeamento de Maps.

```
Comparator<AbstractMap.Entry<Integer, TreeSet<AbstractMap.Entry<String, Integer>>>> comp2 = Map.Entry.comparingByKey();
TreeSet<AbstractMap.Entry<Integer, TreeSet<AbstractMap.Entry<String, Integer>>>> filtroFinal = new TreeSet<>(comp2);
```

Comparators e TreeSets de difícil compreensão.

```
for (Map.Entry<String, Map<String, Integer>> pair : resultsReviews.entrySet()){
    TreeSet<AbstractMap.Entry<String, Integer>> aux = new TreeSet<>(comp);
    aux.addAll(pair.getValue().entrySet());
    filtro.put(pair.getKey(), aux);
}
```

Ordenação de resultados.

Resultados

Nesta secção apresentam-se testes de tempo de execução, uma vez que não se consegui realizar a parte do projeto correspondente aos testes de desempenho. Os resultados apresentados foram obtidos numa máquina com as seguintes características:

Processador: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz

Gráfica: Intel(R) UHD Graphics 630

RAM: 16 GB (15,8 GB utilizável)

SSD: 910 GB (762 GB disponíveis)

- Estatísticas (parte 1)
  Tempo de execução: 2.6376604 s

- Estatísticas (parte 2)
  Tempo de execução: 6.1653704 s

- Query 1
  Tempo de execução: 1.4530989 s

- Query 2
  Input: 1 (janeiro) de 2005
  Tempo de execução: 0.8416259 s
  Input: 6 (junho) de 2015
  Tempo de execução: 0.9856192 s
  Input: 12 (dezembro) de 2018
  Tempo de execução: 0.9820681 s

- Query 3
  Input: RNm_RWkcd02Li2mKPRe7Eg (user_id)
  Tempo de execução: 1.0089475 s
  Input: QpsbLV8dNFI9cxhrBGA0Rw (user_id)
  Tempo de execução: 1.007715 s
  Input: ZlMtgglrpQBmKAqR-idjWA (user_id)
  Tempo de execução: 0.9723455 s

- Query 4
  Input: AW_dMex_BXFzgBJFxAjDuQ (business_id)
  Tempo de execução: 1.0341559 s
  Input: t2xsi7qIN0iwGiOyCn7k8Q (business_id)
  Tempo de execução: 1.6674316 s
  Input: i9BDFBYcl_PGqrLbQUdMvg (business_id)
  Tempo de execução: 1.1166113 s

- Query 5
  Input: RNm_RWkcd02Li2mKPRe7Eg (user_id)
  Tempo de execução: 1.0215805 s
  Input: QpsbLV8dNFI9cxhrBGA0Rw (user_id)
  Tempo de execução: 1.0313099 s
  Input: ZlMtgglrpQBmKAqR-idjWA (user_id)
  Tempo de execução: 0.9891906 s

- Query 6
  Input: 5
  Tempo de execução: 1.8024128 s
  Input: 25
  Tempo de execução: 1.7203171 s
  Input: 625
  Tempo de execução: 1.6439197 s

- Query 7
  Tempo de execução: 1.2748224 s

- Query 8
  Input: 5
  Tempo de execução: 1.7694303 s
  Input: 25
  Tempo de execução: 1.5368358 s
  Input: 625
  Tempo de execução: 1.2762836 s

- Query 9
  Input: AW_dMex_BXFzgBJFxAjDuQ (business_id)
  Input: 5
  Tempo de execução: 2.9186046 s
  Input: AW_dMex_BXFzgBJFxAjDuQ (business_id)
  Input: 10
  Tempo de execução: 5.2500784 s
  Input: AW_dMex_BXFzgBJFxAjDuQ (business_id)
  Input: 20
  Tempo de execução: 8.6791181 s

- Query 10
  Tempo de execução: 1.6276858 s

Conclusão

Como jeito de conclusão, tendo em conta que não conseguimos acabar todos os objetivos do projeto (medidas de performance do código e das estruturas de dados), sabemos que o nosso projeto podia estar mais completo e para além disso poderia ter um código mais "apresentável" e menos complexo. Contudo, os resultados aqui alcançados estão bem conseguidos.

Aproveitamos para salientar o facto de que é necessário ter uma boa noção dos conceitos composição, agregação, herança, encapsulamento, entres outros, para conseguir respeitar as suas regras e assim construir um programa capaz de evoluir. Cremos que este, para além dos referidos no início do relatório, tenha sido uns dos principais objetivos e desafios do projeto.

Diagrama de Classes

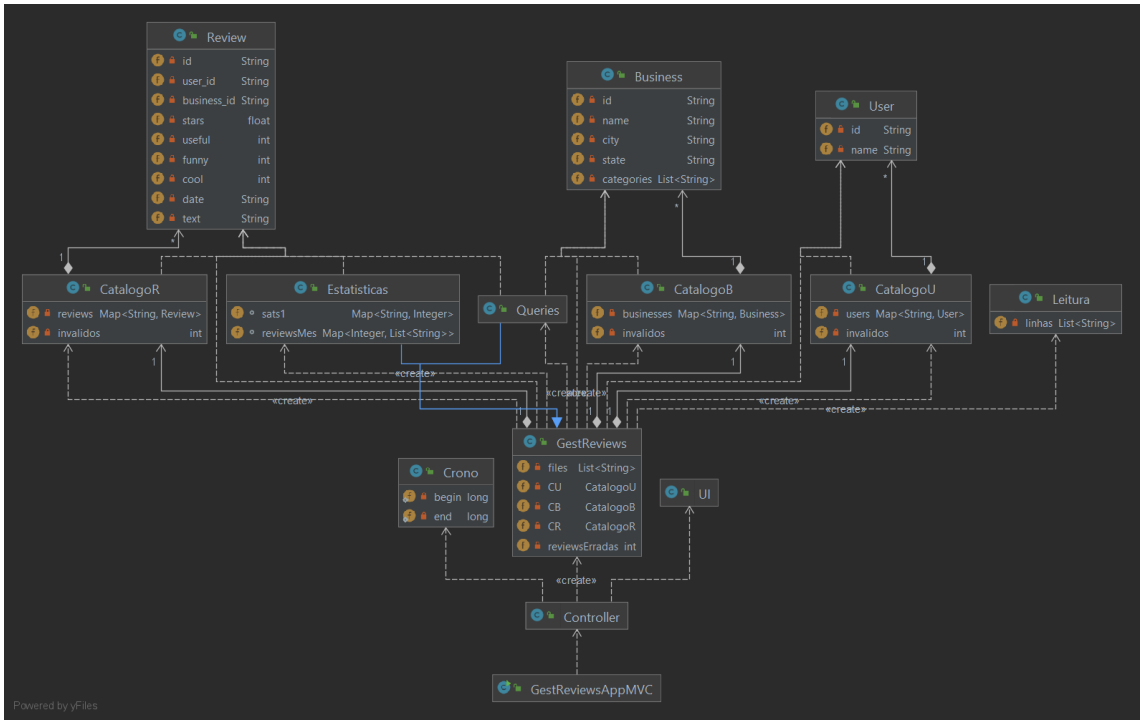

Figura 1: Diagrama de classes do projeto, by **Intellij**