Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Trabalho Prático em *Java*

StackOverflow

LI3

2º Semestre
2017-2018

Grupo 38

A77045 - Ricardo Barros Pereira
A74260 - Luís Miguel da Cunha Lima
A79034 - Rafaela Maria Soares da Silva
A81898 - Hugo Manuel Gomes Nogueira

Junho de 2018
Braga

Conteúdo

1 Introdução

O projeto *"Stack Overflow"* surge no âmbito da segunda fase do projeto de Laboratórios de Informática (LI3), do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Um projeto a ser resolvido na Linguagem *Java*, e este baseia-se no desenvolvimento de um sistema capaz de processar ficheiros XML que guardam um conjunto de informações utilizadas pelo *StackOverflow*. Este sistema tem como objetivo analisar os artigos presentes em *backups* da plataforma do *Stack Overflow*, e na extração de informação útil,para que posteriormente seja possível executar um conjunto de interrogações de forma simples mas de forma eficiente, como por exemplo, saber quais são os utilizadores mais ativos ou quais são os temas mais comuns.

Deste modo, houve a necessidade de pensar em estruturas de dados e algoritmos para a resolução de interrogações no menor tempo possível e de forma eficiente. Numa primeira fase, foi essencial o estudo de *"SAX parser"* como forma de fazer *"parse"* aos *backups* e posteriormente guardar a informação útil nas estruturas e com os tipos que melhor se adequavam. Depois de carregados todos os dados necessários, numa segunda fase foram implementadas as respostas às *"queries"* propostas pelos docentes.

Neste sentido, este relatório é uma tradução de todas as etapas que nos foram propostas de modo a alcançar o objetivo final deste projeto, em que tentamos explicitar da forma mais simples. Temos como objetivo aplicar técnicas e conhecimentos aprendidos e adquiridos nesta Unidade Curricular, em que iremos explicar no decorrer deste relatório.

2 Conceção da Solução

Em grupo, foi proposto que, em conjunto, seriam resolvidas as grandes questões acerca da escolha e implementação da estrutura de dados, da análise e inserção de dados na estrutura. Estando esta primeira fase resolvida, passamos á distribuição das *queries* pelos membros do grupo. Procurou-se, assim, obter uma linha de trabalho clara, eficaz e que nos permitisse obter os resultados esperados.

A estrutura de dados a utilizar pode ter um grande impacto no desenvolvimento das tarefas e na sua eficiência, portanto foi importante desde início definir a forma como tratar e guardar a informação que posteriormente iria ser trabalhada e manipulada pelo programa. Optamos por escolher a criação de estruturas em forma de *Map* para guardar dados relativos as publicações no sistema, Utilizadores que contribuíram com publicações. *Map* são objetos que mapeiam valores para chaves, ou seja, através da chave conseguimos ter acesso ao valor correspondente, sendo que a chave não pode ser repetida ao contrário do valor, mas se caso tiver uma chave repetida, esta entrada é sobrescrita pela última entrada. Esta coleção é utilizada normalmente para guardar, retornar, manipular e transmitir dados agregados de maneira simples, que possui uma arquitetura unificada e criada para permitir que coleções sejam manipuladas de forma independente dos detalhes das suas implementações.

Foram criados dois *HashMaps* de forma a tirar partido da rapidez de inserção e procura de dados deste tipo de Coleção, apesar de não se encontrar ordenado, para guardar informações dos Utilizadores (*ID* do Utilizador). Para os *Posts*, uma vez que devido a grande quantidade das queries que nos são pedidas necessitam de uma certa ordem relativamente a datas, criamos um *TreeMap*, ordenando-o pela Data de publicação de um *Post* de forma a tornar a procura mais eficiente para estes casos.

Para este projeto, utilizados o modelo *MVC(Model-View-Controller)*, o que possibilitou a divisão do projeto em camadas bem definidas tornando a aplicação muito mais modular e estruturada, de forma a tornar possível a reutilização de código e desenvolvimento paralelo de maneira bastante eficiente. Dividimos o nosso código em três *packages*: *common*(classes utilitárias), engine(classes que armazenam os dados das comunidades do *Stack Overflow* e que implementam as *queries*, assim como as funcionalidades relacionadas) e *li3*(definição da interface do sistema de *queries* e código de testes). Estes modulos são completamente independentes, acessíveis somente através da sua API.

2.1 Classes

Para o auxílio da resolução de tudo o que era pedido, resolvemos criar um conjunto de classes, no qual achamos que era importante criar e de forma a que esteja tudo organizado e modulado. Foram elas: Classe *User*, *Post*, *StackOverflow*, *Parser*. Foram ainda criados também exceções para tratamento de erros e comparadores para ajudar nas várias interrogações a que fomos propostos a resolver. Todas as classes têm os métodos de instância habituais.

2.1.1 Classe *User*

Na classe *User*, as variáveis de instância escolhidas foram o identificador do utilizador, assim como o seu nome, o número total de *posts* que fez, a sua *bio* e a sua reputação como utilizador da plataforma do *StackOverflow*, como podemos ver em baixo.

```
private long idU;              //identificador do utilizador
private String name;           //Nome do Utilizador
private numPosts;              //Numero total de posts efetuados pelo  utilizador
private String bio;            //Biografia do Utilizador
private int reputation;        //Reputação do Utilizador
```

2.1.2 Classe *Post*

Na classe *Post*, as variáveis de instância escolhidas foram o identificador do *post*, o nome do título do *post*(aplicável apenas caso o *post* seja uma pergunta), o tipo de *post*(1 se for pergunta, 2 se for resposta),

o identificador do utilizador que o publicou, o *idAnswer*(aplicável caso o *post* seja uma resposta, isto é, o id da pergunta respetiva), a data em que foi publicado, as *tags* que contém, o *score*(*up votes* - *down votes*) e o número de comentários(caso o *post* seja uma resposta).

```
private long idPost;              //identificador do post
private String title;             //Nome título do post
private int typePost;             //Tipo do Post
private long idUser;              //Identificador do utilizador que publicou o post
private long idAnswer;            //Identificador da pergunta(caso resposta)
private LocalDate date;           //Timestamp do post
private String tags;              //Tags do post
private int score;                //Up Votes - Down Votes
private nComments;                //Número de comentarios (só para respostas)
```

2.1.3 Classe *StackOverflow*

Na classe *StackOverflow*, optamos por escolher um *HashMap* e um *TreeMap* como estruturas de dados.

O *HashMap* contém todos os utilizadores do sistema e o outro *TreeMap* contém todos os *posts* ordenados por data do sistema.

Além dos métodos para inserir utilizadores e *posts*, tem todos os métodos associados às interrogações.

```
public class StackOverflow implements TADCommunity {
    // Variáveis de instância
    private Map<Long, User> users;      // Mapeamento entre os users e o seu ID
    private Map<LocalDate, List<Post>> posts;   // Mapeamento entre os posts e a sua data de criaçao
    private Map<Long, Tag> tags;        //Mapeamento entre as tags e o seu ID
```

2.1.4 Classe *Parser*

Decidimos elaborar dois métodos - o *parseUser* e o *parsePost*, de modo a extrair toda a informação necessária e que achamos importante para a resolução de todos os requisitos propostos, de *xml* para *Java*.

Para isto, recorreu-se ao estudo do *"SAX parser"*. Em todos estes métodos, extraímos a informação necessária e inserimos a informação na classe correspondente ao *parser* em questão.

2.1.5 Exceções

Quando se cria programas de computador em Java, há possibilidade de ocorrer erros imprevistos durante a sua execução, esses erros são conhecidos como exceções e podem ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis. No intuito de um eficiente tratamento de erros, decidiu-se criar um conjunto de classes de exceções.

- Classe ***UserInexisteException*** − Exceção que indica que o utilizador não existe.
- Classe ***PostInexistenteException*** − Exceção que indica que o *post* não existe.

2.1.6 *Comparators*

Para comparar utilizadores e *posts* da forma pretendida, decidiu-se criar um conjunto de classes de *Comparators*.

- ***ComparadorDeDatas*** – *Comparator* que compara duas datas.
- ***ComparatorReputacao*** – *Comparator* que compara dois utilizadores, considerando o valor da reputação com cariz decrescente.
- ***ComparatorVotos*** – *Comparator* que compara dois *posts*, considerando o valor dos votos com cariz decrescente.
- ***ComparadorNumPosts*** – *Comparator* que compara dois utilizadores, considerando o número de *posts* com cariz decrescente.

3 Estratégias

3.1 Pergunta 1

Para a interrogação 1, dado o identificador de um *post*, o método deve retornar o título do *post* e o nome do autor da pergunta.

Inicialmente decidimos criar o método *getPost* para conseguirmos extrair facilmente a informação necessária do respetivo *post*, caso este seja uma resposta(pois temos de ir buscar a informação do *post* pergunta). Esta informação resume-se ao tipo de *post*, à obtenção do título do *post* e ao nome do autor.

Percorremos o *TreeMap* de *posts* até encontrarmos o *post* pretendido. De seguida, verificamos se esse *post* é pergunta ou resposta. Caso seja pergunta, obtemos diretamente o título do *post* e o nome do utilizador. Caso contrário, usamos o método *getPost* para extrair a informação da *post* pergunta ao qual a resposta foi efetuada.

3.2 Pergunta 2

Para a interrogação 2, dado um inteiro N, o método deve retornar o top N de utilizadores com maior número de *posts* de sempre.

É criado um *ArrayList* com todos os utilizadores, lista essa que é ordenada usando o *ComparatorNumPosts*, *comparator* definido que compara dois utilizadores usando o seu número de *posts* como método de comparação. Depois de ordenado, é obtido, através desse *ArrayList*, um novo *ArrayList* com tamanho N, que contém os N utilizadores com maior número de *posts*. Por último, é criado um *ArrayList* de *longs* onde irão ser postos os Id's dos respetivos N utilizadores.

3.3 Pergunta 3

O objetivo desta pergunta é bastante simples - queremos obter o número total tanto de perguntas como também o de respostas efetuadas num determinado intervalo de tempo fornecido.

Para alcançar este objetivo, percorremos o *TreeMap* de *posts* e verificamos se este *Post* se encontra no intervalo tempo pretendido. Se sim verificamos se o *Post* é pergunta ou resposta, incrementando o valore respetivo.

3.4 Pergunta 4

Na interrogação 4, queremos todas as perguntas que contenham uma determinada *Tag* fornecida dentro de um intervalo de tempo. Ora, a resolução passou por percorrer a estrutura dos *Posts* e verificar a respetiva data e o tipo de *Post*. Se este completar estes requisitos e ainda contiver a *Tag* anteriormente passada, é adicionado a uma lista final com todos os *Ids* das perguntas.

3.5 Pergunta 5

Para esta pergunta 5, queremos obter a informação do perfil(*short bio*) de um utilizador e dos seus últimos dez *posts*.

Para conseguirmos obter os últimos dez *posts*, recorremos ao método *ordenaArrays*. Este método permite ordenar simultaneamente dois *arrays* - um que contém as datas de todos os *posts* do utilizador pretendido e outro que contém os *ids* respetivos dos *posts*, no intuito de no fim extrair os *ids* dos dez últimos *posts*, ordenados por cronologia inversa.

3.6 Pergunta 6

O objetivo desta pergunta é mostrar as N respostas com melhor classificação dentro de um intervalo de tempo arbitrário.

Para isso, recorremos ao método *getPostsRespostas*, que num determinado período obtém todos os *posts* que são resposta. De seguida, através do *ComparatorVotos*, falado e explicado em cima, ordenamos

os *posts*(resposta) por *score* decrescente. Por fim, basta obter os *ids* dos N primeiros *posts*(resposta) obtidos.

3.7 Pergunta 7

O objetivo desta pergunta é devolver os *IDs* das N perguntas com mais respostas efetuadas. Inicialmente, a abordagem foi, ao mesmo tempo que ia sendo feito o *Parser*, ir atualizando um variável relativa aos *Posts* que ia contando o número de respostas que a respetiva pergunta teria, ou seja, sempre que fosse feito parse de uma respostas em que tinha um determinado *Id* como pergunta, iríamos a essa pergunta e incrementar a variável de contagem de respostas.

Esta estratégia não se tornou viável uma vez que o *parse* iria demorar demasiado tempo, uma vez que iria percorrer a estrutura dos *Posts* tantas vezes quantas respostas havia no *backup* de exemplo.

A nova estratégia passa por inicialmente, percorrer a estrutura dos *posts* e guardar num *Map*(com chave *Id* do *Post* e valor igual número de respostas em que neste caso ainda 0) todas as perguntas dentro do intervalo de tempo pretendido. De seguida, percorrer novamente a estrutura dos *Posts*, mas desta vez toda a estrutura sem restrição de datas, e sempre que apareça uma resposta a uma pergunta que esteja contida no *Map* anterior criado, o valor dessa chave é incrementado. Finalmente baste ordenar as entradas do *Map* pelos valores das chaves e retornar as primeiros N entradas, ou seja, os *IDs* com mais respostas em forma de uma lista de *IDs*.

3.8 Pergunta 8

Para esta próxima pergunta o objetivo passava por retornar uma lista com os *IDs* de N perguntas cujos títulos contenham uma determinada palavra fornecida.

A estratégia de resolução passou por recolher todos os *Posts* que continham a palavra anteriormente falada para que em seguida sejam criados dois *ArrayLists*, um com os *IDs* dos *Posts* e outro com as respetivas datas. No fim estes dois *Arrays* são ordenados de forma decrescente utilizando o método *ordenaArray* que ordena simultaneamente os dois *Arrays* conforme a data. É retornado os N primeiros *IDs* do *Array* de *IDs* criado e preenchido anteriormente.

3.9 Pergunta 9

Dados os IDs de dois utilizadores, o objetivo é devolver as últimas N perguntas (por cronologia inversa) em que participaram os dois utilizadores específicos. Para resolver esta interrogação, guardamos todos os *Posts* respetivos aos dois utilizadores fornecidos em dois *ArrayLists* diferentes. Por fim, são comparados valores dos dois *Arrays* e sempre que um dos *IDs* esteja presente em ambos, é adicionado a um *ArrayList* auxiliar.

Como forma de melhorar a performance, é verificado de entre ambos os *ArrayList* o mais pequeno para que seja percorrido e ver se cada um dos *IDs* presentes neste *Array* está presente no outro. Se sim é adicionado.

3.10 Pergunta 10

Esta interrogação tem como objetivo obter a melhor resposta de uma respetiva pergunta, conforme uma fórmula de cálculo específica.

Para isto, é percorrida toda a estrutura dos *posts* e verificado se o tipo de *post* é do tipo resposta e se o *id* da pergunta é igual ao *id* fornecido. Se sim, é aplicado a fórmula. No fim, a melhor resposta é aquela que obtiver um maior valor conforme o *score* da resposta, a reputação do utilizador, o seu número de votos e de comentários.

$$(Scr \times 0.45) + (Rep \times 0.25) + (Vot \times 0.2) + (Comt \times 0.1)$$

3.11 Pergunta 11

Relativamente a esta pergunta o grupo sentiu algumas dificuldades na sua resolução, tendo criado mesmo estrutura para guardar os dados das *Tags* e ainda alguns métodos auxiliares, no entanto, acabamos por não conseguir o resultado esperado de modo a não apresentarmos a resolução da *querie*.

4 Conclusão

Tendo em conta o objetivo do projeto, a criação de um sistema que permitisse analisar a imensa quantidade de dados do *Stack Overflow*, surgiram algumas dificuldades que tivemos de ultrapassar. Entre elas, melhorar o tempo de execução do *load*, encontrar uma forma de converter a *string* da data em *LocalDate* e encontrar uma forma de ordenar um *TreeMap* por valor e não por chave. Para além destas dificuldades que foram surgindo durante a realização do projeto, tivemos também algumas dificuldades na resolução de algumas *queries* mais complexas, acabando mesmo por não conseguir obter os resultados esperados para a *querie* 11.

A implementação deste sistema permitiu não só o desenvolvimento das capacidades de raciocínio e programação, mas também contribuiu para alargar o nosso conhecimento relativamente ao uso de bibliotecas.

Em suma, o resultado foi positivo. A implementação deste sistema foi conseguida e todo o trabalho foi recompensado com o resultado, o que leva a que o grupo considere que realizou um trabalho bastante bom, uma vez que compreendeu e implementou os conceitos e metodologias aqui faladas.