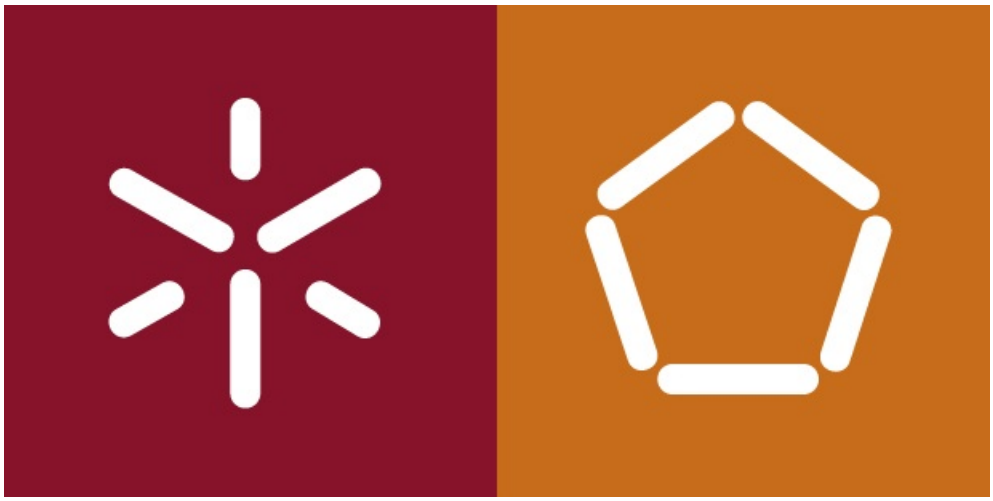
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Relatório - Laboratórios de Informática III

Grupo 19

Trabalho realizado por:	*Número*
Adriana Gonçalves	A75119
Marco Gonçalves	A75480
Ricardo Canela	A74568

Conteúdo

1 Introdução

Este projeto foi-nos solicitado pelos docentes da unidade curricular Laboratórios de Informática III e, esta fase, tem como principal objetivo a realização de um projeto em Java sobre o Stack Overflow com diretrizes semelhantes à fase anterior desenvolvida em C. Através do processamento dos metadados do Stack Overflow, o programa terá de ler e guardar todas as informções necessárias para responder ás interrogações propostas. O Stack Overflow é, atualmente, uma das comunidades de perguntas e respostas mais utilizadas em todo o mundo e, devido à grande utilização desta plataforma e de forma a obter informações como o top N utilizadores com maior número de posts de sempre, o número total de posts num determinado intervalo de tempo, o título de um post e o nome do respetivo autor (dado o id do post) ou a informação do seu perfil pessoal (short bio), como iremos ver mais à frente, é necessário o desenvolvimento de um código legivel, bem estruturado e a utilização de estruturas para fluxo dos respetivos dados. Neste relatório, iremos ainda apresentar e fundamentar as opções de desenho e desempenho do trabalho realizado bem como eventuais limitações/superações dos objetivos propostos.

2 Descrição do enunciado do projeto

O projeto tem como principal objetivo responder a um total de onze interrogações sobre a plataforma de perguntas e respostas Stack Overflow. E para este efeito deve ser desenvolvido um programa em Java bem estruturado e a executar no menor tempo possível de maneira a obter-se uma melhor qualidade de resultados. As informações relativas aos posts, users e votes desta plataforma num determinado instante podem ser acedidos através de ficheiros XML que fornecem toda essa informação. Depois de uma análise detalhada ao enunciado, reparamos que apenas os ficheiros que contêm as tags, users e os posts são os únicos que contêm informações relevantes para uma correta resposta ás querys propostas.

3 Classes/Estruturas

Neste capítulo iremos abordar as classes que utilizamos no desenvolvimento do nosso projeto bem como de todas as suas instâncias e estruturas de dados que suportam as funcionalidades das mesmas.

3.1 TCD - Tipo concreto de dados

O tipo concreto de dados (TCD) contém todas as estruturas principais que foram definidas para a realização do projeto, nomeadamente um HashMap para as tags, para os users e para os posts e uma TreeMap para os days. Um HashMap é uma estrutura de dados que possui a vantagem de associar keys a valores. O seu objetivo é, a partir de uma key, fazer uma pesquisa rápida e obter o valor desejado. Por outro lado, uma TreeMap tem a vantagem de responder a queries relacionadas com intervalos de tempo de forma mais mais rápida e eficaz, daí termos optado por esse tipo de estrutura para os Day.

3.2 Tag

Nesta classe estão definidas as instâncias para as informações necessárias para cada Tag de um post. Assim, esta classe terá como variáveis de instância o id de uma tag, uma string que contém uma determinada tag e um inteiro referente ao nº de vezes que uma dada tag foi usada. Nos métodos desta classe estão incluídos os métodos mais comuns: get's. set's e equals.

```
private long id;
private String tag;
private int n_used;
```

- **id:** contém o id de uma tag.
- **tag:** indica a tag pretendida.
- **n_used:** indica o número de vezes que uma dada tag foi utilizada.

3.3 User

Nesta classe estão definidas as intâncias para as informações necessárias para cada User. Assim, esta classe terá como variáveis de intância o id do user, o seu DisplayName, uma short bio, a

reputação do user, o nº de posts que este fez e uma lista desses mesmos posts. Nos métodos desta classe estão incluídos os métodos mais comuns: get's. set's e equals.

```
private long id;
private String name;
private String aboutMe;
private long reputation;
private int Nposts;
private ArrayList<Post> posts;
```

- **id:** contém o id de um user.
- **name:** indica o DisplayName de um user.
- **aboutMe:** contém a short bio de uma user.
- **reputation:** indica a reputação de um user.
- **Nposts:** indica o número de posts que um user fez.
- **posts:** Lista de posts que um user fez.

3.4 Post

Nesta classe estão definidas as instâncias para as informações necessárias para cada Post. Esta classe servirá como modelo para as outras duas classes que dela dependem evitando assim repetição de variáveis e métodos. Nesta caso, foram criadas as classes Quest e Answer que herdam a informação que está na super classe (Post). Esta classe tem como variáveis de instância o id, data de criação e score de um post, o id do user que elaborou o post e o número de comentários que o mesmo tem. Nos métodos desta classe estão incluídos os métodos mais comuns: get's, set's e equals.

```
private long id;
private LocalDate creationDate;
private int score;
private long ownerUserId;
private int comment_c;
```

- **id:** contém o id de um post.
- **creationDate:** data de criação de um post.
- **score:** score de um post.
- **ownerUserId:** id de um user que elaborou um determinado post.
- **comment_c:** número de comentários que um determinado post tem.

3.5 Answer

Esta classe corresponde a uma Answer, uma "especificação"da classe Post onde estarão todos os dados herdados dessa super classe e mais alguns. A esta classe foram acrecentadas diversas variáveis de instância para poderem ser guardadas informações acerca do id da quest à qual uma answer responde e uma variável que indica a qualidade de uma resposta (avarage). Nos métodos desta classe estão incluídos os métodos mais comuns: get's, set's e equals.

```
private long parentId;
private float average;
```

- **parentId:** contem o id da quest à qual uma answer responde.
- **average:** é uma variável que indica a qualidade de uma resposta.

3.6 Quest

Esta classe corresponde a uma Quest onde estarão também todos os dados herdados da classe Post. A esta classe foram acrecentadas diversas variáveis de instância para poderem ser guardadas informações acerca do título e tag de uma pergunta, o número de answers que uma quest tem e uma lista de answers de uma determinada quest. Nos métodos desta classe estão incluídos os métodos mais comuns: get's, set's e equals.

```
private String title;
private String tags;
private int answer_c;
private ArrayList<Long> answerList;
```

- **title:** Título da pergunta.
- **tags:** Tag da pergunta.
- **answer_c:** Número de answer que uma quest tem.
- **answerList:** Lista de answers de uma quest.

3.7 Day

Nesta classe estão definidas as intâncias para as informações necessárias para cada Day. Assim, esta classe terá como variáveis de intância um dia, uma lista de post de um determinado dia, o número de quests e answers num determinado dia. Nos métodos desta classe estão incluídos os métodos mais comuns: get's. set's e equals.

```
private LocalDate data;
private Map<Long,Post> posts;
private int nQuest;
private int nAnswer;
```

- **data:** Data de um dia.

- **posts:** Lista de posts de um determinado dia.
- **nQuest:** Número de quests num dia.
- **nAnswer:** Número de answers num dia.

3.8 ComparatorNPosts

```
public class ComparatorNPosts implements Comparator<User>{
public int compare(User a, User b){
```

- **Comparator<User>:** Classe a ser implementada no ComparatorNposts.
- **User a:** User a para a comparação do número de posts efetuados.
- **User b:** User b para a comparação do número de posts efetuados.

3.9 ComparatorDate

```
public class ComparatorDate implements Comparator<Post>{
public int compare (Post p1, Post p2){
```

- **Comparator<Post>:** Classe a ser implementada no ComparatorDate.
- **Post p1:** Post p1 para comparação de data de criação.
- **Post p2:** Post p2 para comparação de data de criação.

3.10 ComparatorAverage

```
public class ComparatorAverage implements Comparator<Answer>{
    public int compare (Answer p1, Answer p2){
```

- **Comparator<Answer>:** Classe a ser implementada no ComparatorAverage.
- **Answer p1:** Answer p2 para comparação da average.
- **Answer p2:** Answer p2 para comparação da average.

3.11 ComparatorDateDay

```
public class ComparatorDateDay implements Comparator<Day>{
public int compare (Day da1, Day da2){
```

- **Comparator<Day>:** Classe a ser implementada no ComparatorDateDay.
- **Day da1:** Day 1 para comparação de data.
- **Day da2:** Day 2 para comparação de data.

3.12 ComparatorDatePost

```
public class ComparatorDatePost implements Comparator<Post>{
    public int compare (Post p1, Post p2){
```

- **Comparator<Post>:** Classe a ser implementada no ComparatorDatePost.
- **Post p1:** Post p1 para comparação de data de criação do post.
- **Post p2:** Post p2 para comparação de data de criação do post.

3.13 ComparatorScorePost

```
public class ComparatorScorePost implements Comparator<Post>{
    public int compare (Post p1, Post p2){
```

- **Comparator<Post>:** Classe a ser implementada no ComparatorScorePost.
- **Post p1:** Post p1 para comparação do score do post.
- **Post p2:** Post p2 para comparação do score do post.

3.14 Parser

Sendo o parser uma componente essencial no bom funcionamento do projecto, é necessário decidir a melhor solução possível. Assim, na existência de vários tipos de parser, optamos pelo StaxParser para implementação do projeto, pois é bom para ser utilizado em grandes documentos não consumindo tanta memória como no caso de outros tipos de parsers. Podemos também processar o documento de dados de forma linear e trabalha muito bem com stream que é utilizado no fluxo de dados no projeto. Ao longo do desenvolvimento do projeto apercebemo-nos que apenas temos a necessidade de parsar três tipos de dados: as Tags, os Users e os Posts. Assim, teremos três funções auxiliares com o objetivo de carregar as nossas estruturas. A função parseTags irá percorrer o ficheiro xml e por cada row, ele criará uma tag com a informação respetiva e insere-a na HashMap das tags. Para os Users, o processo é análogo ao das Tags. Para os Post, a parsePost terá a função de criar um novo post (com o id, uma data, score, userId, parentid, tittle, tags, nAnswer e nComment) e inseri-lo em todas as estruturas que lhe dizem respeito. Começa por verificar se o post é do tipo pergunta ou resposta, no primeiro caso é inicializado uma quest com os valores recolhidos e é inserida na HashMap das quest e numa TreeMap de um dos dias (caso o dia com a data do post nao exista é criado um novo dia com a mesma), para além de incrementar a variavel Nposts do user que o criou e de o adicionar à lista de respostas desse mesmo user. Caso o post seja uma resposta é tomado o mesmo procedimento para o caso das perguntas.

4 Queries

Após todas as classes e estruturas estarem bem definidas e os dados devidamente armazenados, estão assim reunidas as condições necessárias para começarmos a responder ás onze queries referidas no enunciado.

4.1 Info-From-Post

Para a resolução desta interrogação, teremos inicialmente de saber se o id do post fornecido é relativo às quest ou às answer. Caso se trate de uma quest, basta aceder à Hash das quests e retirar o título do post correspondente e o ownerUserId do user que elaborou a quest. Seguidamente, acedemos à estrutura dos users e, através do ownerUserId do post, obtemos o DisplayName do user que criou a quest. Caso o post seja relativo a uma answer, teremos de obter o parentId do post sendo que este retornará o id da quest à qual esta answer responde. Depois disto, a forma de obter o título do post e o nome do utilizador autor do post retorna-se de forma análoga ao caso das quests.

4.2 Top-Most-Active

Para a resolução desta interrogação é necessário receber um inteiro N correspondente ao número de utilizadores pretendidos com mais posts de sempre (tanto perguntas como respostas). Para isso, basta iterar a estrutura dos User e utilizar a variável Nposts para criar uma lista ligada ordenada por este fator. Esta é guardada na estrutura de dados principal, pois quando a execução da query for efetuada mais do que uma vez será somente necessário truncar a lista pelos N elementos necessários.

4.3 Total-Posts

—

4.4 Questions-With-Tag

—

4.5 Get-User-Info

Para a resolução desta interrogação, com o id do user fornecido e de forma a obter a short bio e os Id's dos seus últimos 10 posts, é apenas necessário aceder à estrutura User e retornar a bio (aboutMe) e, posteriormente ordenar a ArrayList dos posts desse user pela data, e retornar os 10 mais recentes.

4.6 Most-Voted-Answers

Para a resolução desta interrogação é necessário receber duas datas (Date begin e Date end), e na estrutura dos Day's, aceder ao HashMap das post (Map<Long,Post> posts) de cada dia desse intervalo de forma a obtermos o score de cada resposta e efetuar a criação de uma lista ligada de answers ordenada por esse fator. No final a função retorna os Id's das N respostas com mais votos.

4.7 Most-Answered-Questions

—

4.8 Contains-Word

Para a resolução desta interrogação, dada uma palavra, será necessário aceder à estrutura das Quest e ver em cada título das quest se essa palavra está contida. No final, retorna uma lista com os id's das N perguntas cujos títulos a contenham, ordenados por cronologia inversa.

4.9 Both-Participated

Para a resolução desta interrogação, dados dois id's de dois users e um inteiro N, é necessário aceder à estrutura dos Users, e ao id de cada user procurar os posts em que cada um desses utilizadores participaram (HashMap<Long,Post> posts). Depois de carregar as hash com longs (id's das quest) haverá uma remoção dos id's dos post ficando apenas aqueles em que ambos participaram, a hash dos id's dos posts será transformada numa lista sendo esta posteriormente ordenada. No final, retorna a lista de id's das N perguntas em que ambos os user participaram.

4.10 Better-Answer

Para a resolução desta interrogação, dado o id de uma pergunta, será necessário aceder à estrutura das Quest e recolher a ArrayList<Long> answerList (lista das answer de uma determinada quest) e verificar de qual das respostas é a melhor. Para isso basta verificar a average de todas as respostas a essa pergunta e retornar a que tiver melhor média.

4.11 Most-Used-Best-Rep

—

5 Conclusão

Com a conclusão deste projeto foi possível perceber os diferentes aspetos entre as duas linguagens de programação utilizadas para a realização deste trabalho (C e Java), bem como as mais valias entre uma e outra. Após a finalização do projeto, podemos concluir que apesar de em Java termos achado que o modo como se implementa e como se responde às queries seja relativamente mais simples, em C, estimativamente, o programa realiza essas ações com muita mais rapidez. Posto isto, verificamos uma grande diferença na leitura e armazenamento dos programas entre as duas fases do projeto. Assim, concluímos que ao longo da realização deste projeto, fomos confrontados com algumas dificuladades, sendo que a maior foi decidir que caminhos tomar para melhor responder ao pedido.