Universidade do Minho

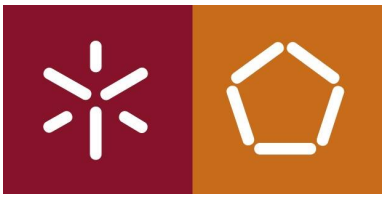
Mestrado Integrado em Engenharia Informática

Sistemas Operativos

Projecto em Java

Autores:
João Gomes (A74033)
Tiago Fraga (A74092)
Ricardo Silva (A60995)

11 de Junho de 2017



Conteúdo

Capítulo 1

Introdução

Nesta segundo projeto da unidade curricular de Laboratórios de Informática III foi-nos pedido que implementasse-mos o mesmo conjunto de requisitos que no projeto anterior,só que desta vez em Java. Este segundo projeto tem como objetivo a aplicação dos conhecimentos obtidos recentemente acerca desta linguagem. Após a realização de um semelhante projeto para a primeira parte, porém em C, também será um dos objetivos da elaboração deste novo projeto apercebermos-nos da vantagem de Java quanto a C quando se trata de criação de estruturas de dados bem como armazenamento de grandes quantidades de dados. Para este projeto tivemos uma ajuda inicial do corpo docente,fornecendo-nos uma *interface*, uma classe *main* além de um executavel .sh de modo a facilitar o teste do nosso programa.

Capítulo 2

Estrutura de Dados

O motivo de nós termos escolhido estruturas *Hash* para o armazenamento de dados é devido a oferecerem um desempenho de tempo constante para as operações básicas (adicionar, remover, contém e tamanho). Apesar de não ser garantida qualquer ordem aos seus elementos isso não é relevante para os dados armazenados. Isto porque, quando é necessário aceder a eles, ou precisamos de aceder a todos ou precisamos de aceder a dados com uma certa propriedade, dependendo da *querie*.

Capítulo 3

Classes Implementadas

Neste capitulo, vamos abordar e explicar a implementação das classes utilizadas para resolver o problema. As classes criadas de forma a resolver o problema foram: *Artigo,Contribuidor,Revisao,Artigos,Contribuidores, Revisoes* e *Parser*.

3.1 Classe Artigo,Contribuidor,Revisao

3.1.1 Artigo

Nesta *Classe* definimos o objeto Artigo que contêm as variáveis de instância,os construtores, os métodos de instância e ainda os métodos *equals*, *toClone* e *toString*.

- Variáveis de Instância: As variáveis de instância criadas são as seguintes:

```
// Titulo do Artigo
private String titulo;

// Id do Artigo
private long idTitulo;

// numero de caracteres do texto de cada Artigo
private long nCaracteres;

// numero de palavras do texto de cada Artigo
private long nPalavras;

// Objecto da Classe Revisoes
private Revisoes revisoes;
```

  Como podemos observar esta classe contem um objeto de outra classe, que vamos explicar posteriormente.

- Construtores: Para esta classe resolvemos implementar os três construtores usuais, visto facilitarem o modo como é criado o Artigo.

```
public Artigo();

public Artigo(String titulo, long idTitulo, long nCaracteres, long nPalavras);

public Artigo(Artigo a);
```

  Com estes construtores implementados, somos capazes de criar um Artigo vazio, um Artigo com base nos parâmetros recebidos e ainda um Artigo com base noutro Artigo recebido.

- Métodos de instância:

  Depois de pensadas as variáveis de instância que iria ter esta classe, tivemos de criar os *get's* e os *setter's* de modo a podemos obter os valores de cada artigo.

- Equals,toClone,toString:

  Criamos ainda os métodos para comparação, para cópia e para transformar Artigo numa string.

3.1.2 Contribuidor

Esta classe contêm a definição do objeto Contribuidor que contêm tal como o Artigo as variáveis de instância,os construtores ,métodos de instância, bem como o *equals*,*toClone* e *toString*.

- Variáveis de instância Para a classe Contribuidor optamos por ter as seguintes variáveis:

```
// Username do Contribuidor
private String username;

// Id do Contribuidor
private long idUsername;

// Numero de contribuições
private long quantidade;
```

- Construtores

  Mais uma vez criamos os três construtores usuais.

```
public Contribuidor();

public Contribuidor(String username, long idUsername, long quantidade);

public Contribuidor(Contribuidor c);
```

- Métodos de instância

  Para as variáveis de instância criadas, foram criados os respetivos *getter's* e *setter's*.

- Equals,toClone,toString

  Tal como na classe Artigo criamos os métodos de comparação,cópia e transformação da classe numa *String*.

3.1.3 Revisao

Esta classe contêm a definição do objecto *Revisao* que contêm tal como o Artigo e Contribuidor as variáveis de instância, métodos de instância,os construtores, bem como o *equals*,*toClone* e *toString*.

- Variáveis de instância:

  As variáveis para a classe *Revisao* são as seguintes:

```
// Id da Revisão
private long idRevisao;

// TimeStamp da Revisão
private String timeStamp;
```

- Construtores Construtores criados:

```
public Revisao();

public Revisao(long idRevisao, String timeStamp);

public Revisao(Revisao r);
```

- Métodos de instância Para as variáveis de instância criadas desta classe, foram criados os respetivos *getter's* e *setter's*.
- Equals,toClone,toString

  Tal como na classe Artigo e Contribuidor criamos os métodos de comparação,cópia e transformação da classe numa *String*.

Estas três Classes são classes que servem de base aos objetos criados posteriormente, são os elementos mais simples do nosso programa.

3.2 Classe Artigos

Esta classe torna-se mais complexa que as anteriores visto conter uma das estruturas utilizadas que é *HashMap*, esta classe vai conter todos os métodos de manipulação desta estrutura.

Nesta classe resolvemos colocar como variáveis de instância o numero total de artigos,numero total de artigos únicos, numero de revisões e uma *HashMap* que contêm todos os artigos e tem como *key* o *id* do artigo e como *value* o Artigo.

```
// Numero total de Artigos
private long total;

// Numero total de Artigos únicos
private long unicos;

// Numero de revisões
private long allRevisions;

// Estrutura com todos os Artigos
private HashMap<Long, Artigo> artigosPorTitulo;
```

Nesta classe e nas próximas classes foram desenvolvidos só os construtores que achamos necessários para a nossa implementação. Esta classe contêm os respetivos *getter's* e *setter's* para cada variável de instância criada.

3.2.1 Métodos Relevantes

- insereArtigo:

  Este método é o responsável por inserir um dado Artigo na nossa *HashMap* bem como incrementar os valores da nossas variáveis de instância. Recebe ainda um *id* da revisão e uma data da mesma, de modo a acrescentar uma nova revisão ou caso necessário criar uma nova.

```
public boolean insereArtigo(Artigo artigo, long idRevisao, String data){
    boolean inseriu = false;
    boolean articleExits = this.artigosPorTitulo.containsKey(artigo.getIdTitulo());

    // Se o artigo não existir cria
    if(articleExits == false){
        Revisao revisao = new Revisao(idRevisao,data);
        Revisoes mapRevisoes = new Revisoes();
        mapRevisoes.insereRevisao(revisao);
```

```
                artigo.insereMapRevisaoArtigo(mapRevisoes);
                this.artigosPorTitulo.put(artigo.getIdTitulo(), artigo);
                this.total++;
                this.unicos++;
                this.allRevisions++;
                inseriu = false;
            // caso exista altera/acrescenta informação
            }else{
                Artigo artg = this.artigosPorTitulo.get(artigo.getIdTitulo());
                artg.setTitulo(artigo.getTitulo());
                if(artigo.getnCaracteres() > artg.getnCaracteres()){
                    artg.setnCaracteres(artigo.getnCaracteres());
                }
                if(artigo.getnPalavras() > artg.getnPalavras()){
                    artg.setnPalavras(artigo.getnPalavras());
                }
                Revisao revisao = new Revisao(idRevisao,data);
                inseriu = artg.getRevisoes().insereRevisao(revisao);
                if(inseriu == false){
                    this.allRevisions++;
                }
                this.total++;
            }
            return inseriu;
        }
```

- getTitulosComPrefixo:

  Este método é invocado como auxiliar do método que implementa a *querie* que devolve a lista de titulos com determinado prefixo como parâmetro.

```
        public List<String> getTitulosComPrefixo(String prefix) {
            List<String> result = new ArrayList<>();

            for(Artigo artigo : this.artigosPorTitulo.values()) {
                if(artigo.getTitulo().startsWith(prefix)){
                result.add(artigo.getTitulo());
                }
            }

            return result;
        }
```

3.3 Classe Contribuidores

Esta classe contêm um *HasMap* com todos os contribuidores existentes, em que a *key* é o *id* do Contribuidor e o *value* é o Contribuidor.Contêm ainda como variável de instância o numero total de contribuidores.

```
// Numero total de Contribuidores
private long quantos;

//HashMap com todos os Contribuidores
private HashMap<Long, Contribuidor> contribuidores;
```

Mais uma vez implementamos os construtores necessários,*getter's* e *setter's* e ainda os métodos *equals*, *toClone* e *toString*.

3.3.1 Métodos Relevantes

- insereContribuidor: Este método é responsável por inserir um contribuidor caso ele não exista, ou então atualizar os dados relativos ao Contribuidor dado como parâmetro.

```
public void insereContribuidor(Contribuidor contribuidor){

    boolean contributorExits = this.contribuidores.
                                containsKey(contribuidor.getIdUsername());

    // Caso não exista insere
    if(contributorExits == false){
        this.contribuidores.put(contribuidor.getIdUsername(), contribuidor);

        this.quantos++;
    }
    // Caso contrário atualiza
    else{
        Contribuidor contribuidorFromHash= this.contribuidores.
                                            get(contribuidor.getIdUsername());

        long result = contribuidorFromHash.getQuantidade() + 1;
        contribuidorFromHash.setQuantidade(result);
    }
}
```

3.4 Classe Revisoes

Esta classe faz parte da classe Artigo como tínhamos visto anteriormente, contêm o numero total de revisões seguindo a ideologia das anteriores, e ainda um *HashMap* com todas as revisões em que a *key* é o *id* de revisão e o *value* é a Revisão.

```
// Numero total de Revisões
private long quantos;

// HashMap com todas as Revisões
private HashMap<Long,Revisao> revisoes;
```

Tal como nas classes apresentadas anteriormente,foram implementados os construtores necessários e os métodos de comparação.cópia e transformação da classe numa *String*.

3.4.1 Métodos Relevantes

- insereRevisao: Este método vai inserir uma revisão dada como parâmetro na estrutura caso essa revisão não exista, devolvendo um *boolean* para indicar que essa revisão existe.

```
 public boolean insereRevisao(Revisao revisao){
    boolean revisaoExits = this.revisoes.containsKey(revisao.getIdRevisao());

    // Caso não exista a Revisão
    if(!revisaoExits){
        this.revisoes.put(revisao.getIdRevisao(), revisao);
        this.quantos++;
    }
    return revisaoExits;
}
```

Capítulo 4

Parser

Neste capítulo vamos abordar uma das classes mais importantes de todo o nosso programa que é o *Parser* de *libXML* utilizado.

Para implementar o *parser* capaz de retirar os valores pretendidos dos *snapshots* fornecidos, utilizamos como base o *StAX*. Utilizamos este tipo como base, visto o outro *parser* indicado pelo corpo docente, apresenta-se uma grande desvantagem, que era carregar todos os dados para memória (*DOM*). Visto a grande quantidade de dados que tínhamos de trabalhar, o nosso programa usando o *DOM parser* iria apresentar grandes problemas ao nível da velocidade de processamento.

Esta classe é uma das fundamentais para o funcionamento do nosso programa, visto que sem ela não iríamos conseguir trabalhar os dados fornecidos.

È aqui que retiramos os valores e chamamos os construtores de Artigo, Contribuidor para criar estes objetos e chamamos os métodos responsáveis pela inserção dos mesmos nas respetivas estruturas.

Capítulo 5

QueryEngineImpl

Neste capitulo vamos explicar como foi implementada a classe relativa as *queries*. Fazendo referência as soluções idealizadas para as resolver.

5.1 Queries

5.1.1 Querie 1, 2 e 3

- 1 - AllArticles: *Querie* que devolve a quantidade total de Artigos: Este método obtém a resposta à query 1 com bastante facilidade pois o numero total de artigos é incrementado sempre que ocorre uma inserção ou uma atualização. Desta forma a resposta a esta query limita-se a fazer uma leitura do valor de uma variavel da classe Artigos.
- 2 - UniqueArticles: *Querie* que devolve a quantidade total de Artigos únicos, isto é, a quantidade total sem repetição. Este método, tal como o método que responde à query 1, é obtido com facilidade pois sempre que ocorre uma inserção de um novo artigo, isto é, um artigo que ainda não existia no hashmap é incrementada uma variavel que tem o total de artigos unicos. Assim, tal como anteriormente, é obtida a resposta a esta query lendo o valor que se encontra numa variavel da classe Artigos.
- 3 - AllRevisions: *Querie* que devolve a quantidade total de Revisões: Para obter a resposta a esta query, tal como nas anteriores, sempre que há uma inserção de um artigo, neste caso um artigo que já existe no hashmap, é incrementado o valor de uma variavel que possui o total de revisões. Assim sendo, como nas anteriores, a resposta é dada com base na leitura de uma variavel.

5.1.2 Querie 4

Para responder à query 4, isto é, obter os 10 maiores contribuidores existentes, foi necessário obter a partir do hashmap de contribuidores todos os contribuidores. Na classe Contribuidor, que é o objecto que representa um contribuidor, existe uma variavel que conta para quantos artigos aquele contribuidor contribuiu. Essa variavel é incrementada sempre que é inserido um novo artigo pertencente a esse contribuidor. Assim, depois de obtidos todos os contribuidores, foi utilizada uma stream e foram ordenados com base num comparador ComparatorContribuidor que implementa um comparador de Contribuidor recorrendo ao metodo sorted. No encadeamento desta operação foram limitados os resultados a 10 unidades (para obter o top 10) e por fim convertidos para uma lista com os resultados. O código que ilustra a parte principal do algoritmo desta query encontra-se de seguida:

```
tmp = allContributors.values().stream()
    .sorted(new ComparatorContribuidor())
    .limit(10)
    .collect(Collectors.toList());
```

5.1.3 Querie 5

Nesta *querie* é nos pedido que retornemos o nome do contribuidor a que pertence o *id* dado como parâmetro. Como o *id* é a *key* do nosso *HasMap* de contribuidores e possuímos um método dentro da classe Contribuidores que nos retorna o *HashMap* pretendido, basta ir buscar a estrutura e com o método *get* presente na *API* desta estrutura,conseguimos retirar o Contribuidor com aquele *id*, bastando depois retornar o seu nome.

5.1.4 Querie 6

Para a resolução desta *querie* utilizamos uma *stream* para percorrer o *HashMap* de Artigos.

```
tmp = allArticles.values().stream()
      .sorted(new ComparatorArtigoCaracteres())
      .limit(20)
      .collect(Collectors.toList());
```

Como podemos observar percorremos o *HashMap* e organizamos os artigos de modo a que estejam pela ordem decrescente de numero de caracteres, para isso cria-mos uma classe auxiliar que trata da comparação.Depois de ordenada a estrutura retiramos os primeiros vinte elementos e guardamos tudo numa *List* de Artigo temporária. Para retornarmos os *id's* de artigos num *ArrayList* percorremos a *List* criada e adicionamos a estrutura a devolver o *id* do Artigo guardado.

5.1.5 Querie 7

Nesta *querie* é pretendido que seja devolvido o titulo de determinado Artigo, recebendo o *id* como parâmetro, logo tal como na *querie* o nosso *HashMap* de artigos, na classe Artigos possuí como *key* o *id* do artigo, logo basta ir buscar ao nosso *HashMap* o artigo em questão e retornar o seu titulo.

5.1.6 Querie 8

Para responder à query 8, isto é, obter os N artigos com mais palavras, foi necessário obter a partir do hashmap de artigos todos os artigos. Na classe Artigo, que é o objeto que representa um artigo, existe uma variável com o numero de palavras do artigo. Essa variável é preenchida na instanciação do objeto recorrendo a uma função auxiliar que faz a contagem das palavras do artigo. Assim, depois de obtidos todos os artigos, foi utilizada uma stream e com base num comparador *ComparatorArtigoPalavras* que implementa um comparador de Artigos com base no numero de palavras e recorrendo ao método *sorted* foram ordenados. No encadeamento desta operação foram limitados os resultados a N unidades (para obter o top N - este N é passado por argumento para o método) e por fim convertidos para uma lista com os resultados. O código que ilustra a parte principal do algoritmo desta query encontra-se de seguida:

```
tmp = allArticles.values().stream()
    .sorted(new ComparatorArtigoPalavras())
    .limit(n)
    .collect(Collectors.toList());
```

5.1.7 Querie 9

Para responder à query 9, isto é, todos os titulos de artigos que começam com um determinado prefixo foi necessário obter a partir do hashmap de artigos todos os titulos. O hashmap de artigos possui como chave o identificador do artigo por isso foi necessário utilizar o método getTitulosComPrefixo para obter uma lista com todos os titulos que se encontram na classe Artigos. Depois de obtida essa lista, recorrendo ao método de ordenação sort (como se apresenta no código abaixo) foram ordenados os titulos por ordem alfabética.

```
this.artigos.getTitulosComPrefixo(prefix);
```

De seguida, os titulos foram adicionados a uma lista e devolvidos como resposta à interrogação.

5.1.8 Querie 10

A ultima *querie* pedida é para retornar o valor do *timeStamp* recebendo um *id* de artigo e de revisão, para a resolução desta questão precisamos de ir buscar primeiro o *HashMap* de artigos, depois retirar o artigo pretendido e ir buscar o *HashMap* de revisões presente na classe *Revisoes*, de seguida retiramos a revisão pretendida com o auxilio do *id* que é *key* da estrutura, retornando o valor presente na variável *timeStamp*.

Capítulo 6

Conclusão

Após a conclusão e entrega deste projeto, há aspetos que se destacam quando comparados à mesma fase de desenvolvimento do programa elaborado em C, concebido na primeira parte da unidade curricular.

Uma das dificuldades que mais sentimos a quando da realização do projeto em C foi, sem dúvida, a estruturação do programa – era algo que estava sujeito constantes alterações de modo a facilitar e/ou a levar a uma maior eficiência do código – alterações essas que levam a um grande replaneamento do que já foi implementado, uma vez que se trata duma linguagem em que trabalha sobre a memória.

Porém, nesta segunda fase da unidade curricular, após pensadas as estruturas necessitadas, foi muito mais fácil de as implementar em JAVA, uma vez que dispomos de grandes quantidades de interfaces sob onde trabalhar. Não só limitados a essa vantagem, como uma das grandes vantagens de JAVA é o alheamento do programador quanto a operações na memória, uma vez que esta linguagem contém um sistema de gestão de memória, mais conhecido como *garbage collector*.

Como já referido, uma das grandes dificuldades do trabalho em C foram as constantes alterações da estruturação do programa. Ora neste projeto isso não se verificou, em parte devido à sua semelhança com o trabalho realizado em C, mas também à grande lição que aprendemos na fase anterior, a importância de previamente estudar e planear bem o projeto, antes de saltarmos logo para a escrita do código.