Universidade Lusófona de Humanidades e Tecnologias
Algoritmia e Estruturas de Dados
2022/2023
2º Semestre - 1ª época
v1.0.1

Enunciado do projeto prático

Segunda Parte

Nota prévia

A parte 2 do projeto deve ser uma continuação do trabalho que iniciaram na parte 1. Os testes da parte 1 vão-se manter na parte 2, pelo que quem não passava nesses testes deve agora trabalhar nisso, antes de avançar para as novas funcionalidades.

Devem-se manter os grupos definidos para a primeira parte. Não será permitida a entrada de novos membros nos grupos.

Objectivos

Mantém-se o objetivo da primeira parte - desenvolver um programa capaz de receber e responder a pedidos de informações/estatísticas sobre uma base de dados de músicas e artistas (bandas, músicos, etc.)

Deverão, portanto, continuar o projeto entregue na parte 1.

Organização do trabalho

Este trabalho está dividido em duas partes, sendo ambas de entrega obrigatória.

Este é o enunciado da Segunda Parte.

A não entrega de qualquer uma das partes do projeto implica reprovação na componente prática da disciplina.

Objetivos da Segunda Parte

Nesta segunda parte do projeto pretende-se que os alunos realizem as seguintes tarefas:

- Suporte para o *input* de "perguntas"/comandos que serão feitas à base de dados e para o cálculo e apresentação dos resultados respetivos.
- Suporte para alteração da base de dados em memória
- Otimização do programa para lidar com ficheiros "gigantes", de forma a que as queries sejam respondidas num tempo razoável (alguns segundos, no máximo)

Devem manter o foco da parte 1 em garantir a execução correta do programa mas passam também a ter que garantir que usam os algoritmos e estruturas de dados mais adequadas para que o programa execute rapidamente. Lembrem-se do que aprenderam nas aulas. O processo de otimização deve ser iterativo - comecem por identificar o bottleneck, experimentem várias soluções para o otimizar e façam medições de tempos para decidir a melhor solução.

Realidade / Domínio do Problema

As entidades principais neste problema são as mesmas que existiam na primeira parte do projeto:

- Tema Musical
- Artista

Nesta segunda parte, o artista passa a poder estar associado a uma ou mais etiquetas (tags).

Uma etiqueta é uma String sem espaços que caracteriza um grupo de artistas. Exemplos de etiquetas são "punk", "indie", "foiAoIdolos".

Ficheiros “gigantes”

Nesta segunda parte do projeto, os testes do Drop Project serão realizados com ficheiros “gigantes”, com dados reais de músicas e artistas, retirados de uma base de dados pública.

Para facilitar os testes, estes ficheiros são disponibilizados aos alunos no seguinte endereço: https://github.com/ULHT-AED-2022-23/rock-in-deisi-large-files

Ao utilizarem estes ficheiros, vão-se deparar com 2 tipos de problema que não tiveram na parte 1:

- O programa que desenvolveram na parte 1 vai provavelmente ficar excessivamente lento. A chamada à função loadFiles(...) poderá passar a demorar alguns minutos ou mesmo horas (!). Isto é normal, pois não tiveram (nem tinham que ter) a preocupação de otimizar o vosso código - relembramos que na parte 1 o foco era na correção e não na rapidez de execução.

- Existirão mais situações de linhas inválidas do que as que tinham previsto na parte 1. Estas situações já estavam nos ficheiros originais da base de dados pública, não foram introduzidas pelos professores! E refletem a realidade que vão encontrar na vossa vida profissional: os ficheiros têm muitas vezes dados mal formatados, “lixo”, etc… Quanto mais cedo se habituarem a lidar com isso, melhor!

Queries

Como já foi referido, o programa terá de conseguir responder a perguntas sobre a base de dados (BD) de temas musicais. Essas perguntas são normalmente designadas de queries e seguem um formato padrão.

Por exemplo, poderemos saber as 10 músicas mais dançáveis que foram lançadas entre 1990 e 2000 através desta query:

```
GET_MOST_DANCEABLE 1990 2000 10
```

Que retornará isto:

```
Ice Ice Baby : 1990 : 0.98
Teachers : 1997 : 0.98
Binibi Rocha - Live : 1990 : 0.978
Ice Ice Baby - Radio Edit : 1990 : 0.977
```

```
Barney Theme Song : 1994 : 0.976
Last Night a D.J. Saved My Life : 1991 : 0.968
Pray : 1990 : 0.9670000000000001
Tonite : 1991 : 0.965
Pelo Suelto : 1991 : 0.965
Girlfriend - Single Version : 2000 : 0.963
```

Cada pergunta padrão tem um conjunto de uma ou mais variáveis, que permitem configurar a pergunta.

Desta forma, o *input* do programa será um conjunto de uma ou mais perguntas (e respetivas configurações).

Além disso, existirão comandos que modificam o conteúdo da BD, influenciando o resultado de perguntas posteriores.

Algumas das perguntas padrão são de **implementação obrigatória**. Essas perguntas estão identificadas na tabela com a cor amarela no código da pergunta, e com o texto "(OBG)" na Descrição.

Seguem-se as perguntas padrão que o programa terá de ser capaz de responder.

Código pergunta/comando	**Descrição/Comportamento**	**Variáveis**
COUNT_SONGS_YEAR	Retorna o total de temas musicais que foram lançados no ano X. (OBG)	Uma: o valor de X
COUNT_DUPLICATE_SONGS_YEAR	Retorna o total de temas cujo título está repetido, que foram lançados no ano X.	Uma: o valor de X
GET_SONGS_BY_ARTIST	Retorna as N primeiras músicas associadas ao artista indicado, pela ordem com que aparecem no ficheiro songs.txt. O resultado deve ser uma String contendo várias linhas separadas por \n e respeitando a seguinte sintaxe: "<Nome tema> : <Ano>\n"	Duas: N - número de resultados A - nome do artista

	Caso não hajam resultados (por não existirem músicas para aquele artista ou por o artista não existir), deve retornar “No songs” (OBG)	
GET_MOST_DANCEABLE	Retorna as N músicas mais dançáveis entre os anos X e Y (inclusivé). O resultado deve ser uma String contendo várias linhas separadas por \n e respeitando a seguinte sintaxe: “<Nome tema> : <Ano> : <Dançabilidade>\n” Caso hajam empates, a ordem com que aparecem é indiferente. (OBG)	Três: X - ano início (inclusive) Y - ano fim (inclusive) N - número de resultados
GET_ARTISTS_ONE_SONG	Retorna todos os artistas que num dado período apenas lançaram uma música, ordenados alfabeticamente pelo nome do artista. O resultado deve ser uma String contendo várias linhas separadas por \n e respeitando a seguinte sintaxe: “<Nome artista> | <Nome da música> | <Ano>\n”	Duas: X - ano início (inclusive) Y - ano fim (inclusive) Nota: Caso o ano de início seja igual ou superior ao ano de fim, a query deverá retornar “Invalid period”
GET_TOP_ARTISTS_WITH_SONGS_BETWEEN	Retorna uma lista com os N artistas que mais temas musicais têm, considerando apenas aqueles que têm entre A e B temas (ambos inclusivé). O resultado deve ser uma String contendo várias linhas separadas por \n e respeitando a seguinte sintaxe: “<Nome Artista> <Nr Temas>\n” Os artistas devem ser devolvidos por	Três: Os valores de: ● N ● A ● B

	ordem **decrescente** do número de temas. Caso não existam N resultados, devem ser devolvidos os que existirem. Caso não haja nenhum, deve ser devolvida a string “No results”.	
MOST_FREQUENT_WORDS_IN_ARTIST_NAME	Deve calcular as N palavras que mais vezes aparecem nos nomes dos artistas que têm pelo menos M temas musicais. Apenas devem ser consideradas as palavras que tenham tamanho igual ou superior a L. O resultado deve ser uma String contendo várias linhas separadas por \n, usando a seguinte sintaxe: “<Palavra> <Nr. Ocorrências>\n” As palavras devem ser ordenadas por ordem **crescente** do “<N. Ocorrências>”.	Três: Os valores de: ● N ● M ● L
GET_UNIQUE_TAGS	Deve devolver os nomes de todas as tags que existem na Base de Dados, assim como a quantidade de vezes que cada tag é usada. O resultado deve ser uma String contendo várias linhas separadas por \n, usando a seguinte sintaxe: “<Nome tag> <Nr Ocorrências>\n" A lista deve estar ordenada por ordem **crescente** do ”<Nr. Ocorrências>”. Caso não existam resultados, deve ser devolvida a String “No results”.	
GET_UNIQUE_TAGS_IN_BETWEEN_YEARS	Deve devolver os nomes das tags que existam na base de dados associadas a Artistas que tenham pelo menos uma música entre os dois anos passados como variáveis (ambos inclusivé).	Duas: ● Ano1 ● Ano2

	O resultado deve ser uma String contendo várias linhas separadas por \n, usando a seguinte sintaxe: “<Nome tag> <Nr Ocorrencias>\n” A lista deve estar ordenada por ordem **decrescente** do ”<Nr. Ocorrências>”. Caso não existam resultados, deve ser devolvida a String “No results”.	
GET_RISING_STARS	Retorna os artistas que, cumulativamente: ● têm músicas em todos os anos entre Ano1 e Ano2 (ambos inclusive). ● A popularidade média anual dessas músicas cresceu sempre, de ano para ano. **Ordenação** Os resultados devem ser ordenados pela popularidade média (**arredondada para o inteiro seguinte**) de cada artista nos anos envolvidos na query. Em caso de empate na popularidade média, os resultados empatados devem aparecer por ordem alfabética do nome do artista. Os artistas a retornar devem ser os 15 melhores em termos de popularidade média anual, considerando todos os anos. Caso não existem 15 resultados, devem ser devolvidos os que existirem. **Formato**: uma lista de Strings com a seguinte sintaxe: “<Nome Artista> <=> <Pop média>\n”. Nota: a <Pop média> apresentada deve ser a não arredondada). Caso não existam resultados, deve ser devolvida a String “No results”.	Três: - Ano1 (int) - Ano2 (int) - Ordenação (String) A variável **ordenação** tem dois valores possíveis: ● “ASC” - para ordenação crescente ● “DESC” - para ordenação decrescente

ADD_TAGS	Associa etiquetas/tags a um determinado artista. As tags devem ser associadas em maiúsculas, mesmo que introduzidas em minúsculas. Não podem estar associadas tags duplicadas ao mesmo artista - se se tentar associar a mesma tag mais do que uma vez, deve ser ignorado. Deve retornar a seguinte linha: `<ARTIST> | <TAG>,<TAG>,...` com todas as tags atualmente associadas ao artista respetivo. Note-se que o artista podia já ter anteriormente tags associadas, logo poderão ser apresentadas mais tags do que as que foram introduzidas no input. A ordem das tags não precisa de ser a mesma com que foram inseridas. Caso o artista não exista, deve retornar "Inexistent artist" (OBG)	<ARTIST>;<TAG1>;<TAG2>;… <ARTIST> é o nome do artista Nota: As tags não podem ter espaços

REMOVE_TAGS	Este comando deve remover tags de um determinado artista.
	
	Deve retornar a seguinte linha:
	`<ARTIST> | <TAG>,<TAG>,...`
	com todas as tags atualmente associadas ao artista respetivo.
	
	Caso o artista tenha ficado sem tags, deve retornar:
	`<ARTIST> | No tags`
	
	Caso o artista não exista, deve retornar "Inexistent artist"
	(OBG)
		<ARTIST>;<TAG1>;<TAG2>;…
		
		Onde
		
		<ARTIST> é o nome do artista.

GET_ARTISTS_FOR_TAG	Retorna a lista de artistas associadas a uma certa TAG, ordenados alfabeticamente. O resultado deve ser uma String com uma única linha, com o nome dos artistas separados pelo carater ';' (sem espaços).	Uma: a TAG

	Exemplo: `Nirvana;Metallica` Caso não haja nenhum, deve retornar "No results". (OBG)	

No anexo I, há alguns exemplos de utilização destas perguntas.

Componente de criatividade

Os alunos estão convidados a inventar a sua própria pergunta, dentro das seguintes restrições:

1. A pergunta deve ter um nome elucidativo daquilo que faz (em Inglês) e usar um formato similar ao das perguntas "oficiais"
2. A pergunta tem obrigatoriamente que receber um parâmetro ano (inteiro) que condicionará o resultado. Isto é, se a query der os mesmos resultados independente do ano passado como parâmetro, não será aceite.
3. A pergunta tem obrigatoriamente que retornar um conjunto de coisas (músicas, artistas, etc.) sob o formato de String com várias linhas
4. A pergunta tem que fazer sentido e ser útil. Por exemplo, a pergunta "Quais as músicas cujo id começa pelo algarismo 0." não fornece informação muito útil… 🤦
5. A pergunta tem que ser descrita no ficheiro README.txt e demonstrada no vídeo, além da pergunta não-obrigatória (ver seção Artefatos a entregar) caso contrário não será contabilizada.
6. Tem que existir (pelo menos) um teste unitário sobre essa query

Serão avaliadas a **originalidade**, a **utilidade** e a **complexidade** de implementação da pergunta.

Requisitos técnicos

Devem descrever a vossa query criativa num ficheiro README.txt, que deverá estar na raiz do projeto. A descrição deve incluir o nome da query, uma descrição da mesma e um exemplo real (com os dados dos ficheiros gigantes), num formato similar a este:

```
Nome: GET_LOUDEST_ARTISTS <YEAR>
Descrição: Apresenta os 5 artistas que participaram nas músicas
mais barulhentas de um certo ano, ordenados do mais barulhento para
o menos barulhento.
Exemplo:
GET_LOUDEST_ARTISTS 2000
Deftones
Korn
Rage Against the Machine
System Of a Down
SlipKnot
```

Devem implementar um teste unitário chamado `testCreativeQuery()` com pelo menos 2 execuções da query com parâmetros diferentes. Exemplo:

```
public void testCreativeQuery() {
    // ler ficheiros
   String result = Main.perguntar(“GET_STUFF 2000”);
   assertEquals(“34 blabla\n56 bleble”, result);

   result = Main.perguntar("GET_STUFF 1995");
   assertEquals("32 olaola\n21 oleole\n89 olioli", result);
}
```

Notem que este teste de exemplo depende da leitura de ficheiros. Na secção Testes Unitários Automáticos, é dada mais informação sobre isso.

Dados de entrada (*Input*) do programa

O *input* do programa é constituído por duas componentes:

- Três ficheiros de texto - as sintaxes desses ficheiros de texto **são as mesmas** que foram apresentadas na primeira parte do projeto.
- Interação manual (teclado) *com o utilizador* - explicado na subsecção seguinte.

Mantêm-se todas as situações inválidas da parte 1, excepto a possibilidade de haver espaços extra entre os símbolos separadores dos artistas. Por exemplo, linhas como esta deixam de

existir:

```
1 @  [ 'The National']
2 @ [   'Nirvana']
3 @ "['Radiohead' , 'Foo Fighters'] "
```

No entanto, surgem outras situações que não estavam a ser testadas na parte 1:

- O nome de alguns artistas contém uma plica. Por exemplo, a banda “Jane’s Addiction”. Estes casos são problemáticos pois originariam linhas assim:
  `2l7iDl470hxtX7FkzAA3wz @ ['Jane's Addiction']`
  em que o programa poderia confundir-se e achar que a banda se chamava “Jane” e o resto era um erro.
  Por essa razão, os autores dos ficheiros originais decidiram rodear estes nomes de duplas aspas. Esta linha fica então assim:
  `2l7iDl470hxtX7FkzAA3wz @ "[""Jane's Addiction""]".`
  O programa deve processar a linha de forma a remover as aspas duplas, ficando apenas `Jane’s Addiction`.
- O nome de alguns artistas contém vírgulas. Por exemplo, existe um compositor chamado “Joseph Hellmesberger, Jr.”. Mais uma vez, este caso é problemático pois origina linhas assim:
  `5ToyzS2fIgdPg3k @ "['Joseph Hellmesberger, Jr.', 'Chris Thielemann']"`
  O programa pode confundir-se e achar que aquela vírgula é um separador, identificando apenas o primeiro artista como sendo o “Joseph Hellmesberger”.
  O vosso programa deve ter em conta estes casos e incluir a vírgula no nome. As únicas vírgulas que são consideradas separadores são as que estão fora das plicas.
- Existem linhas em que vários artistas estão dentro das mesmas plicas, separados por vírgula. Possivelmente isto é um erro na criação dos ficheiros originais mas teremos quer lidar com ela. Dada a regra explicada no ponto anterior, esses casos devem ser considerados válidos.
  Exemplo:
  `7nuW6IlOP5kf @ "['Vanessa Bell Armstrong, Patti Austin, Bernie K.’]"`
  Esta linha deve ser considerada válida e criar um único artista com o nome “`Vanessa Bell Armstrong, Patti Austin, Bernie K.`”

O processamento destes casos é bastante complexo e sai um pouco do âmbito da disciplina. Decidimos manter estes casos porque achamos que não devemos desvirtuar os ficheiros originais (os casos já lá estavam, não foram inventados por nós). No mundo profissional, terão que lidar com todo o tipo de ficheiros portanto isto acaba por ser uma boa preparação.

No entanto, sejam espertos e usem as ferramentas à vossa disposição, nomeadamente o chatgpt (ver secção “Utilização do chatgpt”).

Interação manual (teclado) com o utilizador - `Standard Input`

O programa a desenvolver deverá correr de forma interativa.

Ao arrancar o programa, o mesmo deverá ficar "à escuta" de input, que será fornecido via teclado / `standard input` (`stdin`). No Anexo II é apresentado um exemplo de código Java para tratar da interação com o utilizador.

O *input* será uma String, podendo ocorrer uma das três situações seguintes:

- O *input* representa uma *query* válida (constituída pelo código da pergunta e pelo número correto de variáveis);
- O *input* é a `String` "EXIT";
- O *input* representa outra coisa - sendo, neste caso, inválido.

O que fazer em cada situação...

Se o *input* for a `String` "EXIT", o programa dos alunos deve terminar imediatamente.

(Pode assumir que o "EXIT" será sempre feito com todas as letras maiúsculas.)

Se o *input* for uma query válida, o programa dos alunos deve:

1. calcular a resposta à pergunta;
2. apresentar no ecrã essa mesma resposta;
3. apresentar no ecrã o tempo (em *ms*) que demorou a calcular a resposta;
4. ficar "à escuta" da próxima *query*.

Se o *input* representa outra coisa qualquer, o programa deve:

1. apresentar no ecrã a mensagem de erro "Illegal command. Try again"
2. ficar "à escuta" da próxima *query*.

Um exemplo deste esquema de input é apresentado no **Anexo II** deste enunciado.

Nota sobre a validade das *queries*:

Se uma *query* tiver como primeira componente um código de uma pergunta padrão conhecida, então é garantido que o resto da *query* será também válido (formato, variáveis, etc.).

Dados de saída (Output) do programa

O *output* do programa serão as respostas às *queries*, tal como foi descrito na secção anterior do enunciado.

Este *output* deve ser feito para o ecrã.

Restrições Técnicas e Comportamento a implementar

Existem algumas restrições técnicas que terão de ser obrigatoriamente cumpridas pelo código dos alunos. Trabalhos que não cumpram estas restrições serão avaliados com a nota **zero**. Algumas destas restrições implicam adaptações do código feito na primeira parte do trabalho.

As restrições são as seguintes:

- Devem ser mantidas todas as funções pedidas na parte 1. O package também se mantém o mesmo e todas as classes criadas na parte 2 devem fazer parte desse package.

- Deve existir uma nova classe chamada `Query` com os atributos:
  - `name` - String com o nome da query (ex: “COUNT_SONGS_YEAR”)
  - `args` - array de strings com os argumentos da query
    Exemplo: se a query fôr `GET_SONGS_BY_ARTIST 3 Pearl Jam`, então o `args` terá o valor `{ “3”, “Pearl Jam” }`

- Deve existir uma nova classe chamada `QueryResult` com os atributos:
  - `result` - String contendo o resultado (pode conter várias linhas, através do “\n”)
  - `time` - long contendo o tempo de execução da query em milisegundos

- Devem acrescentar as seguintes funções:

```
static Query parseCommand(String command)
static ArrayList parseMultipleArtists(String line)
static QueryResult execute(String command)
```

- A função `parseCommand` processa um comando introduzido pelo teclado e retorna um objeto da classe Query com os atributos devidamente preenchidos. Exemplo de comando: “GET_SONGS_BY_ARTIST 3 Pearl Jam”.

Se o comando introduzido não fôr válido, deve retornar `null`.

- A função `parseMultipleArtists(String line)` processa parte de uma linha do ficheiro song_artists.txt que contenha múltiplos artistas e retorna uma lista com os nomes desses artistas. Esta função deve ser gerada pela chatgpt (ver secção "Utilização do chatgpt").

- A função `execute(String command)` deve executar o comando que lhe for passado no argumento command e devolver o respetivo resultado que será do tipo QueryResult.
    - Nota: o comando fornecido pode não ser válido.
    - A função dos alunos terá de validar o comando e, no caso de o mesmo ser inválido, deve retornar null.
    - No entanto, na versão interativa (quando o utilizador introduz comandos através do teclado), devem apresentar a mensagem "Illegal command. Try again", tal como explicado na secção "Interação manual com o teclado"

- Nesta parte passa a ser permitida a utilização da classe Map e das suas variantes.

- Tendo em conta o que aprenderam nas aulas, devem dividir o vosso projeto em vários ficheiros e manter a classe Main (e respetiva função main) o mais pequena possível.

- Não é permitida a utilização de streams na implementação do projeto.

- Para fazerem ordenações, podem usar o Collections.sort(). No documento "Cenas da API do Java que dão jeito em AED" disponibilizado no Moodle têm mais informação sobre como o fazer.

Utilização do chatgpt

Mantém-se a recomendação na utilização do chatgpt, explicada na parte 1 - usem-no para vos apoiar, não para vos substituir.

Função `parseCommand`

Incentivamos que usem o chatgpt para implementar a função `parseCommand`, não porque seja difícil de implementar mas porque é trabalhosa (tem que lidar com todas as queries e respectivos formatos) e preferimos que o vosso tempo seja gasto noutras partes mais interessantes do projeto.

A função encontra-se descrita na secção “Restrições Técnicas e Comportamento a implementar”.

Devem incluir no ficheiro README.txt, toda a interação que tiveram até obter a função que finalmente vos faz passar o teste `parseCommand`.

Função `parseMultipleArtists`

Voltamos a incentivar (ainda com mais força na parte 2) que o usem para o processamento do ficheiro de artistas, agora que foram introduzidos casos mais complexos (ver secção “Dados de entrada (Input) do programa”). E mais uma vez, essa utilização será bonificada, se devidamente documentada.

Devem então, pedir ao chatgpt que implemente a função obrigatória `parseMultipleArtists` que recebe uma String (uma linha do ficheiro) e retorne uma lista de nomes de artista. Ele trabalha melhor se lhe fornecerem vários exemplos.
Por exemplo:
`parseMultipleArtists(“[‘aaa’, ’bbb’]”)` deve retornar uma lista com “aaa” e “bbb”.

Claro que este é o caso simples, convém também dar-lhe exemplos para os casos mais problemáticos.
Por exemplo:
`parseMultipleArtists(“"['a,aa', 'bbb', 'ccc']"”)` deve retornar uma lista com “a,aa” e “bbb” e “ccc”.

Quanto mais exemplos derem melhor.

Devem incluir no ficheiro README.txt, toda a interação que tiveram até obter a função que finalmente vos faz passar o teste `parseMultipleArtists`. O código desse teste é disponibilizado no anexo III.

IMPORTANTE: Para ambas as situações, devem incluir toda a interação. Na parte 1, alguns de vocês forneceram apenas parte da interação.

Testes Unitários Automáticos

Nesta componente, os alunos devem continuar o trabalho iniciado na parte 1 e acrescentar **casos de teste unitários automáticos** para as novas funcionalidades, nomeadamente a função que executa as queries.

Deverá existir pelo menos um teste para cada query.

Para que os testes às queries sejam realistas, devem incluir todo o processamento que é feito sobre os dados.

Ou seja, cada teste deve:

- Chamar o `loadFiles(...)` com a pasta "test-files" ou uma sub-pasta da mesma. A sub-pasta pode ser uma boa estratégia para terem vários conjuntos de ficheiros de teste (para diferentes casos/queries)
- Chamar vários `execute(...)` para uma certa query com diferentes parâmetros. Se chamarem apenas com um parâmetro (ex: apenas para o ano 2000), o que vos garante que vai funcionar bem para outros parâmetros?
- Cada um dos resultados retornados pelo execute(...) deve ser validado, usando assert's (assertNotNull, assertEquals, assertArrayEquals, etc..)

Segue-se um exemplo de um teste para a query GET_SONGS_BY_ARTIST:

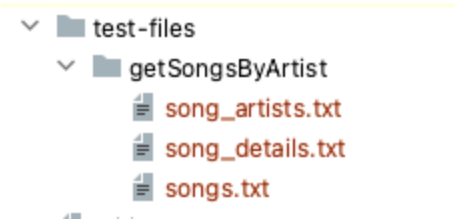
```
@Test
public void getSongsByArtist_OBG() {

   Main.loadFiles(new File("test-files/getSongsByArtist"));

   QueryResult queryResult = Main.execute("GET_SONGS_BY_ARTIST 3 Queen");
   assertNotNull(queryResult);
   assertNotNull(queryResult.result);
   String[] resultParts = queryResult.result.split("\n");
   assertEquals(3, resultParts.length);
   assertArrayEquals(new String[] {
           "song1 : 1991",
           "song3 : 1992",
           "song5 : 1997",
   }, resultParts);

   queryResult = Main.execute("GET_SONGS_BY_ARTIST 1 Nirvana");
   assertNotNull(queryResult);
   assertNotNull(queryResult.result);
   resultParts = queryResult.result.split("\n");
   assertEquals(1, resultParts.length);
   assertArrayEquals(new String[] {
           "song2 : 1991"
   }, resultParts);
}
```

Neste caso, estamos a verificar 2 casos de teste:

- GET_SONGS_BY_ARTIST 3 Queen
- GET_SONGS_BY_ARTIST 1 Nirvana

Para cada um deles, verificamos o resultado, transformando-o num array com as linhas retornadas e usando o `assertArrayEquals` para compará-lo com o resultado que estamos à espera.

Notem que este exemplo pressupõe a existência de uma sub-pasta “getSongsByArtist” na pasta “test-files”, dentro do qual estão os 3 ficheiros de teste:



Neste caso, usou-se o seguinte conteúdo para os ficheiros de teste:

songs.txt

```
1 @ song1 @ 1991
2 @ song2 @ 1991
3 @ song3 @ 1992
4 @ song4 @ 1991
5 @ song5 @ 1997
```

song_artists.txt

```
1 @ ['Queen']
2 @ ['Nirvana']
3 @ ['Queen']
4 @ "['Rage Against The Machine', 'Nirvana']"
5 @ "['Queen', 'Nirvana']"
```

song_details.txt

```
1 @ 349467 @ 1 @ 33 @ 0.912 @ 0.153 @ -11.659
2 @ 293457 @ 1 @ 39 @ 0.372 @ 0.544 @ -5.231
3 @ 238947 @ 2 @ 22 @ 0.798 @ 0.402 @ -9.112
4 @ 983457 @ 3 @ 16 @ 0.612 @ 0.334 @ -6.432
5 @ 672394 @ 3 @ 27 @ 0.923 @ 0.269 @ -8.011
```

Vídeo demonstrativo

Para demonstrarem o correto funcionamento do vosso programa, deverão gravar um vídeo que mostre a interação com o mesmo assim como a descrição do código que o implementa.

Devem respeitar as seguintes regras:

- Cada grupo deve escolher uma query não obrigatória que tenha implementado e passado nos testes. Por exemplo, a query COUNT_SONGS **não** pode ser escolhida, pois é obrigatória
- O vídeo deve mostrar 2 interações com essa query, passando-lhe parâmetros diferentes.
- O vídeo deve explicar sucintamente a forma como foi implementada essa query, apresentando no ecrã o código e uma narração dos próprios alunos do grupo. Deve mostrar o código da função ou funções envolvidas nessa query, assim como as estruturas de dados que possibilitam essa query.
- O vídeo deve começar pela frase “Vou explicar como implementei a query XXX” e tem que ter entre 60 e 90 segundos. Vídeos que não cumpram esta regra terão zero.
- Caso o grupo tenha implementado uma query extra (componente de criatividade), também deve incluir no vídeo uma demonstração e explicação da mesma. Nesse caso, a duração máxima do vídeo passa para 120 segundos.
- Quanto mais detalhada fôr a explicação melhor. Devem explicar não só **como** é que implementaram mas também **porque** é que implementaram dessa forma (frases do género: “Podia ter usado X ou Y mas decidi usar Z porque …”). Se no vídeo se limitarem a ler o código (“Aqui faço um ciclo de 0 a 10, aqui atribuo 1 à variável número, etc.), terão uma nota muito baixa.
- O código apresentado tem que ser legível e o áudio tem que ser percetível. Vídeos que não cumpram esta regra terão zero.
- Recomendamos que façam vários *takes* para garantir que a vossa comunicação é percetível e sem hesitações ou pausas.
- O vídeo deve ser carregado para o youtube e configurado como “**unlisted**” de forma a não aparecer nos resultados de pesquisa. O título do vídeo deve incluir o nome e número dos alunos que constituem o grupo (ex: “aed-antonio-silva-21701234-joana-almeida-21704567”).
- O link para o vídeo deve ser incluído no ficheiro README.txt.

Entrega e Avaliação

Artefactos a entregar

A entrega deve ser feita usando um ficheiro comprimido (.zip) dentro do qual se encontrem:

- Todo o código necessário para compilar e executar o projecto;

- Um ficheiro de texto (chamado AUTHORS.txt) contendo os nomes e números de aluno dos membros do grupo).

- Um ficheiro de texto (chamado README.txt) contendo:
    - Link para o vídeo no youtube (ver secção “Vídeo”)
    - Descrição da query criativa, caso a tenham implementado (ver secção “Query criativa”)
    - Registo da interação completa com o chatgpt para obter a função `parseMultipleArtists` e a função `parseCommand`, tal como descrito na secção “Utilização do chatgpt”
- Ficheiros de teste usados pelos vossos testes unitários (ver secção Testes Automáticos Unitários)

IMPORTANTE: Não devem incluir no vosso .zip os ficheiros “gigantes”, pois isso irá aumentar o tamanho do ficheiro e ultrapassará o limite de upload do DP.

Formato de entrega

O ficheiro .zip deve seguir a estrutura que se indica de seguida:

```
AUTHORS.txt  (contém linhas NUMERO_ALUNO;NOME_ALUNO, uma por aluno do grupo)
README.txt   (contendo link para o vídeo e descrição da query criativa)

… (outros ficheiros)
+ src
|---+ pt
|------+ ulusofona
|---------+ aed
|------------+ rockindeisi2023
|--------------- Main.java
|--------------- ...   (outros ficheiros java do projecto)
|--------------- TestXXX.java
|--------------- ... (outras classes de testes)+ test-files
+ test-files
|--- songs.txt
|--- song_artists.txt
|--- song_details.txt
|--- ...
```

Reparem que esta estrutura corresponde ao *package* obrigatório previamente indicado.

Nota importante: Os ficheiros que vão incluir na pasta “test-files” devem ser ficheiros pequenos inventados por vocês e não os ficheiros gigantes fornecidos. Esses serão automaticamente incluídos pelo Drop Project e usados nos testes dos professores.

Como entregar - Drop Project

A entrega do projeto deverá ser feita usando o sistema Drop Project (DP), situado no URL seguinte:

https://deisi.ulusofona.pt/drop-project/upload/aed-2223-projeto-p2

Será considerada para avaliação a melhor submissão que o grupo faça.

Podem também entregar usando o plugin para Intellij que foi divulgado nas aulas práticas.

Filosofia do uso do DP

Pelas mesmas razões descritas na parte 1, cada grupo apenas pode fazer uma submissão a cada 20 minutos.

Prazo de entrega

A data limite de entrega é o **dia 12 de Junho de 2023 (segunda-feira), pelas 23h00m** (hora de Lisboa, Portugal). Não serão aceites trabalhos após essa data e hora.

As defesas decorrerão entre os dias 14 e 16 de Junho. A data e hora exata serão anunciadas mais próximo da data.

Avaliação

Mantêm-se as regras da primeira parte, às quais se acrescenta:

- Na **nota final** do projeto existe a nota mínima de 9.5 (nove e meio) valores.

- Para garantir que os alunos que vão à defesa do projecto têm um conjunto mínimo de funcionalidade implementada para que o projecto possa ser defendido com sucesso, alguns dos testes vão ser **obrigatórios**.
- Estes testes estarão identificados com o sufixo "_OBG".

- Os alunos que entreguem projectos que não passem **todos os testes obrigatórios** serão reprovados em primeira época.

Cotações da segunda parte:

Item	**Descrição**	**Cotação (valores)**
Componente funcional	Através de um conjunto (bateria) de testes unitários automáticos (aplicados via DropProject). Esta cotação será dividida pelo número total de testes no DP.	12
Avaliação do Professor	Os Professores irão realizar testes extra das funcionalidades a implementar.	2,5
Eficiência	Calculado em função do somatório de tempo que o programa dos alunos demora a calcular as respostas para as *queries* padrão dos testes automáticos. Será publicado um leaderboard que mostra os tempos de cada grupo (anonimizado) ordenado do menor para o maior, para promover uma competição saudável entre os grupos. Ver nota(1) abaixo.	1,5
Testes automáticos	Ver secção “Testes Unitários Automáticos” Por caso de teste considera-se uma função/método anotada com `@Test` que cumpra os requisitos indicados na secção respectiva.	1
Criatividade	Pergunta extra (ver secção “Componente de criatividade”)	1
Vídeo	Vídeo exemplificativo da forma como implementaram uma query não	1

	obrigatória e da query que inventarem (criatividade). Ver regras na secção "Artefactos a entregar"	
Utilização de chatgpt	Ver secção "Utilização de chatgpt"	1

(1) Seria potencialmente injusto que o tempo de execução de projetos que só implementaram 50% das queries sejam comparados com projetos que implementaram 100% das queries. Por essa razão, serão aplicados os seguintes tetos a este componente:
   (a) Passam todos os testes DP - até 1,5 valores
   (b) Passam mais de 90% dos testes DP - até 1 valores
   (c) Passam entre 80% e 90% dos testes DP - até 0,5 valor
   (d) Abaixo de 80% não têm cotação neste item

Notas importantes:

- O programa submetido pelos alunos tem de compilar e executar no contexto do Drop Project.
- Projetos que não passem as fases de estrutura, de compilação serão considerados como não entregues e terão nota zero.
- Projetos que tenham erros de qualidade de código (checkstyle) terão uma penalização de 3 valores na nota final da parte 2

Cópias

Trabalhos que sejam identificados como cópias serão anulados e os alunos que os submetam terão nota zero em ambas as partes do projeto (quer tenham copiado, quer tenham deixado copiar).

Para evitar situações deste género, recomendamos aos alunos que nunca partilhem ou mostrem o código do seu projeto a pessoas fora do grupo de trabalho.

A decisão sobre se um trabalho é uma cópia cabe exclusivamente aos docentes da unidade curricular.

Outras informações relevantes

Mantêm-se todas as informações dadas na parte 1, mas chamamos novamente a atenção para as seguintes informações:

– Existirá uma defesa presencial e individual do projeto. Durante esta defesa individual, será pedido ao aluno que faça alterações ao código para dar resposta a alterações aos requisitos. Da discussão presencial de cada aluno, resultará uma nota de 0 a 100%, que será aplicada à nota do projeto.

Anexo I - Exemplos de uma sessão interativa

Segue-se o exemplo de uma sessão interativa do programa. O texto de início e fim do programa são meramente ilustrativos.

O *output* do programa é apresentado com a cor preto.

A *input* do utilizador é apresentado com a cor azul e fundo amarelo.

Os resultados e tempos apresentados (em milisegundos - ms) são meramente exemplificativos.

```
Welcome to DEISI Rockstar!
COUNT_SONGS_YEAR 2000
1048
(took 62 ms)

COUNT_DUPLICATE_SONGS_YEAR 2000
18
(took 63 ms)

GET_SONGS_BY_ARTIST 10 Nine Inch Nails
Head Like A Hole : 1989
Terrible Lie : 1989
Closer : 1994
Hurt : 1994
The Perfect Drug : 1997
The Hand That Feeds : 2005
Down In It : 1989
Sin : 1989
Sanctified : 1989
Something I Can Never Have : 1989
(took 95 ms)
```

```
GET_MOST_DANCEABLE 2011 2013 3
Go Girl : 2012 : 0.986
Tag der Befreiung : 2011 : 0.975
Vogelvlucht - Original Mix : 2012 : 0.961
(took 11 ms)

GET_ARTISTS_ONE_SONG 1970 1975
94 East | If You See Me | 1975
A Foot In Coldwater | (Make Me Do) Anything You Want | 1972
Ace Spectrum | I Don't Want to Play Around | 1974
Adalberto Santiago | Descarga Fania - Live | 1971
...
(took 421 ms)

ADD_TAGS Nirvana;Rockalhada
Nirvana | ROCKALHADA
(took 0 ms)

GET_ARTISTS_FOR_TAG Rockalhada
Nirvana
(took 63 ms)

GET_TOP_ARTISTS_WITH_SONGS_BETWEEN 10 5 5
Lindsey Buckingham 5
White Noise Baby Sleep 5
Hoagy Carmichael & His Orchestra 5
...
(took 37 ms)

MOST_FREQUENT_WORDS_IN_ARTIST_NAME 8 10 10
Springfield 3
Montgomery 3
Instrumental 3
Philadelphia 3
Bhattacharya 3
Symphonique 3
Thelonious 5
Philharmonic 7
(took 29 ms)

EXIT
Adeus.
```

Anexo II - Código de suporte para Leitura de Dados a partir do Standard Input

A classe **`Scanner`** do Java, para além de permitir ler ficheiros de texto, também permite fazer leituras diretas do teclado, usando o **`System.in`** (nome que representa o *standard input* - que geralmente corresponde ao teclado).

As funções do `Scanner`, quando usadas para interacção com o **`System.in`**, são bloqueantes - ou seja, que o programa vai estar parado enquanto o utilizador não introduzir o seu *input*.

De seguida apresenta-se uma possível lógica para gestão da interactividade:

```
System.out.println("Welcome to Rock in DEISI!");

if (!loadFiles(new File("."))) {
   System.out.println("Error reading files");
   return;
}

Scanner in = new Scanner(System.in);

String line = in.nextLine();

while (line != null && !line.equals("QUIT")) {
   QueryResult result = execute(line);
   if (result == null) {
       System.out.println("Illegal command. Try again");
   } else {
       System.out.println(result.result);
       System.out.println("(took " + result.time + " ms)");
   }

   line = in.nextLine();
}
```

Anexo III - Código do teste unitário à função parseMultipleArtists

Este é o código exatamente igual àquele que vai ser executado pelo Drop Project, no teste com este nome:

```
@Test
public void parseMultipleArtists() {

   assertArrayEquals(new String[] { "aaa" },
                     Main.parseMultipleArtists("['aaa']").toArray());

   assertArrayEquals(new String[] { "aaa", "bbb" },
                     Main.parseMultipleArtists("['aaa', 'bbb']").toArray());

   assertArrayEquals(new String[] { "a,aa", "bbb" },
                     Main.parseMultipleArtists("['a,aa', 'bbb']").toArray());

   assertArrayEquals(new String[] { "aaa", "bb b" },
                     Main.parseMultipleArtists("['aaa', 'bb b']").toArray());

   assertArrayEquals(new String[] { "a'aa", "bbb" },
                     Main.parseMultipleArtists("[\"\"a'aa\"\", 'bbb']").toArray());

   assertArrayEquals(new String[] { "a' aa", "b, bb", "ccc" },
           Main.parseMultipleArtists("[\"\"a' aa\"\", 'b, bb', 'ccc']").toArray());

   assertArrayEquals(new String[] { "Engelbert Humperdinck", "Herbert von Karajan",
"Bancroft's School Choir", "Elisabeth Grümmer", "Elisabeth Schwarzkopf", "Loughton
High School for Girls' Choir", "Philharmonia Orchestra" },
           Main.parseMultipleArtists("['Engelbert Humperdinck', 'Herbert von
Karajan', \"\"Bancroft's School Choir\"\", 'Elisabeth Grümmer', 'Elisabeth
Schwarzkopf', \"\"Loughton High School for Girls' Choir\"\", 'Philharmonia
Orchestra']").toArray());

   assertArrayEquals(new String[]{"Lin-Manuel Miranda", "'In The Heights' Original
Broadway Company"},
           Main.parseMultipleArtists("['Lin-Manuel Miranda', \"\"'In The Heights'
Original Broadway Company\"\"]").toArray());

}
```