Universidade Lusófona de Humanidades e Tecnologias
LEI / LIG / LEIRT
Algorítmia e Estruturas de Dados
2020/2021
2º Semestre - 1ª época
v1.0.2

Enunciado do projeto prático

🎸 DEISI Rockstar 2021 🎸

Segunda Parte

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:

- Deteção mais sofisticada de linhas inválidas e informação incoerente
- 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

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".

Perguntas padrão

Como já foi referido, o programa terá de conseguir responder a perguntas sobre a base de dados (BD) de temas musicais. Essas perguntas serão baseadas num conjunto de perguntas padrão, que são apresentadas de seguida.

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_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). Exemplo: `Nirvana;Metallica` Caso não haja nenhum, deve retornar "No results". (OBG)	Uma: a TAG
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 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”.	Três: Os valores de: ● N ● A ● B
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 superior a L. O resultado deve ser uma String contendo várias linhas separadas por \n, usando a seguinte sintaxe: “<Palavra> <Nr. Ocorrências>\n”	Três: Os valores de: ● N ● M ● L

	As palavras devem ser ordenadas por ordem **crescente** do “<N. Ocorrências>”.	
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 Ocorrencias>\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). 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”.	Duas: ● Ano1 ● Ano2
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,	Três: - Ano1 (int) - Ano2 (int) - Ordenação (String) A variável **ordenação**

	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".	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.	<ARTIST>;<TAG1>;<TAG2>;… <ARTIST> é o nome do artista Nota: As tags não podem ter espaços

	Caso o artista não exista, deve retornar "Inexistent artist" (OBG)	

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.

CLEANUP	Limpa informação incoerente da BD (apenas em memória). Neste caso, considera-se informação incoerente: todas as músicas que não tenham artistas e/ou detalhes associados e todos os artistas que não estejam associados a nenhuma música. Após correr este comando, estas informações devem ser removidas de todas as estruturas de memória, de forma a que deixem de aparecer em queries posteriores.
	
	Deve retornar a seguinte linha:
	`Musicas removidas: XX;`
	`Artistas removidos: YY`
	(substituindo XX e YY pelos respectivos números)
		n/a

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 dois parâmetros que condicionarão o resultado: um ano (inteiro) e um número que pode corresponder à duração (int), dançabilidade (double), vivacidade (double) ou volume médio (double)
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, as perguntas “Quais as músicas cuja popularidade é igual à dançabilidade” ou “Quais são as músicas cuja duração é negativa?” não fornecem informação muito útil… 🤦
5. A pergunta tem que ser explicada, justificada (isto é, qual a utilidade) e demonstrada no vídeo, além da pergunta não-obrigatória (ver seção Artefatos a entregar) caso contrário não será contabilizada.

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

Requisitos técnicos

- Devem implementar obrigatoriamente duas funções:
  `String getCreativeQuery()`
  Esta função deve retornar o nome da pergunta que inventaram ou null caso não tenham inventado nenhuma.
  `int getTypeOfSecondParameter()`
  Consoante o tipo do 2º parâmetro da query, esta função deve retornar:
  1 - caso seja uma duração
  2 - caso seja um nível de dançabilidade
  3 - caso seja um nível de vivacidade
  4 - caso seja um nível de volume

- 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.execute("GET_STUFF 2000 0.78");
   assertEquals("34 blabla\n56 bleble", result);

   result = Main.execute("GET_STUFF 1995 0.13");
   assertEquals("32 olaola\n21 oleole\n89 olioli", result);
}
```

  Notem que este teste 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.

Relativamente aos ficheiros de *input*, os alunos devem ter cuidados genéricos de garantir que o seu programa lida de forma graciosa às situações problemáticas seguintes:

- Ficheiros não existentes;
- Ficheiros vazios.

Além das situações inválidas previstas na parte 1, devem igualmente detetar situações incoerentes durante a leitura dos ficheiros:

- Artistas associados a músicas cujo id seja inexistente devem ser excluídos da BD (essas linhas devem ser ignoradas)
- Linhas em que um dos artistas esteja vazio deverão ser ignoradas. Exemplo: `2YN4tqTyycSZg @ "['', 'Christian Thielemann']"`
- Detalhes de músicas cujo id seja inexistente devem ser excluídos da BD (essas linhas devem ser ignoradas)
- Campos inválidos, como, por exemplo, a duração de um tema não ser um inteiro (essas linhas devem ser ignoradas)

- [v1.0.2] Linhas cujo id esteja duplicado devem ser ignoradas (ex: se aparecerem 3 linhas com um certo id, apenas a primeira deve ser processada. As outras 2 devem ser ignoradas. Isto é válido quer para o songs.txt, quer para o song_details.txt.

Algumas regras/recomendações adicionais a ter em conta na leitura dos ficheiros:

- Existem linhas em que vários artistas estão dentro das mesmas plicas. Exemplo: `7nuW6IlOP5kf @ "['Vanessa Bell Armstrong, Patti Austin, Bernie K.']` Estas linhas não devem ser consideradas inválidas. Estes casos devem ser tratados como se fossem vários artistas. Ou seja, o exemplo acima deveria dar origem a 3 artistas associados a essa música.
- Quer os nomes das músicas, quer o nome dos artistas poderão estar entre aspas duplas. Ex: `""Black 'N Blue""`. Essas aspas devem ser removidas. Neste caso apenas o nome do artista deve ser guardado: `Black 'N Blue`. Caso esteja entre aspas triplas, devem ser mantidas umas aspas. Ex: `"""Eternal"""` deve ser transformado em `"Eternal"`.

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

O que fazer em cada situação...

Se o *input* for a `String` “KTHXBYE”, o programa dos alunos deve terminar imediatamente.

(Pode assumir que o “KTHXBYE” 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:

- O método `toString()` da classe `Song` passa a devolver uma `String` neste formato:

```
ID | Título | Ano | Duração | Popularidade | Nome(s) do(s) artista(s) (Número de temas do artista)
```

A duração deve estar na notação minutos:segundos (ex: 3:23). Os milissegundos restantes devem ser descartados. Não devem colocar zeros à esquerda dos minutos e dos segundos. Por exemplo, devem colocar 3:5 e não 03:05.

Caso exista mais do que um artista, a informação sobre os vários artistas (nome, nr de temas) deve aparecer separados por " / " (espaço-barra-espaço).

À frente de cada nome de artista deve aparecer, entre parêntesis, o número total de temas desse artista.

Exemplo:

```
7hanhZrUArC9qUerln4jh1 | Fake Empire | 2007 | 3:25 | 49.0 | The National | (10)
```

Neste caso mostra-se um exemplo em que o Tema tem um Artista, e esse Artista tem 10 temas..

Nota: Para garantir que o projeto passa os testes automáticos dos Professores, deverá haver um espaço (e apenas um espaço) à volta de cada separador. Por exemplo:
`"7hanhZrUArC9qUerln4jh1|Fake Empire | 2007 | 3:25 |49.0| The National | (10)"`
não será considerado correto pois não tem espaçamento certo.

- Devem ser mantidas todas as funções pedidas na parte 1
- A função `getParseInfo()` deve agora ter em conta os casos que passaram a ser inválidos na parte 2. Mantém-se o `toString()` da classe `ParseInfo` descrito na parte 1.
- Na parte 2 devem acrescentar a seguinte funções:

```
public static String execute(String command)
public static String getCreativeQuery()
Public static int getTypeOfSecondParameter()
public static String getVideoUrl()
```

- A função `execute(String command)` deve executar o comando que lhe for passado no argumento command e devolver o respetivo resultado.
    - 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 devolver `"Illegal command. Try again"`

- A função `getCreativeQuery()` deve retornar o nome da pergunta que inventaram (ver secção “Componente de Criatividade”)

- A função `getTypeOfSecondParameter()` deve retornar o tipo do segundo parâmetro para a pergunta que inventaram (ver secção “Componente de Criatividade”)

- A função `getVideoUrl()` deve retornar o url completo do vídeo no youtube, por ex: https://www.youtube.com/watch?v=wTFP2CtCqv4 (ver secção “Artefatos a entregar”)

- 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 regex ou 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.

Testes Unitários Automáticos

Nesta componente, os alunos devem implementar **casos de teste unitários automáticos** para garantir a qualidade do seu projeto. Por "caso de teste" entende-se a definição de um método que verifique se, para certo "*input"* é obtido o "*output*" / *resultado esperado*.

Nesta segunda parte, deverão implementar pelo menos um teste para cada query/comando.

Testes vazios ou sem `assert`'s não serão considerados.

Os casos de teste devem ser implementados usando **JUnit 4**, tal como demonstrado nas aulas. Cada caso de teste deve ser implementado num método anotado com `@Test`.

Os testes devem ser implementados em uma ou mais classes com o nome `TestXYZ` (em que `XYZ` é o nome da classe que está a ser testada). Por exemplo, uma classe chamada `ContaBancaria` teria os seus testes numa classe chamada `TestContaBancaria`. As classes de teste devem estar no mesmo package que as classes que estão a ser testadas.

Caso os testes unitários precisem de ler ficheiros (para carregamento inicial das estruturas de memória), esses ficheiros deverão ser obrigatoriamente colocados na pasta **test-files**, como se pode ver na estrutura abaixo:

```
AUTHORS.txt  (contém linhas NUMERO_ALUNO;NOME_ALUNO, uma por aluno do grupo)
+ src
|---+ pt
|------+ ulusofona
|---------+ aed
|------------+ deisiRockstar2021
|--------------- Main.java
|--------------- ...   (outros ficheiros java do projeto)
|--------------- TestXXX.java
+ test-files
|--- ficheiro-de-testes.txt
|--- ...
```

Nota: Embora os testes devam ler os ficheiros que estão na pasta test-files, o programa em si (Main) deverá ler os ficheiros que estão na raiz. Os alunos devem preparar o código para as duas situações.

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).
- 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.

Além disso devem gravar um vídeo, segundo 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 uma explicação da forma como foi implementada essa query, apresentando o ecrã com o IDE e áudio com narração do aluno. 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 retornado pela função `getVideoURL()` explicada na secção “Restrições Técnicas”.

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)
+ src
|---+ pt
|------+ ulusofona
|---------+ aed
|------------+ deisiRockstar2021
|--------------- Main.java
|--------------- ...   (outros ficheiros java do projecto)
|--------------- TestXXX.java
+ test-files
|--- ficheiro-de-testes.txt
|--- ...
```

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

Nota importante: Os ficheiros de *input* devem ser lidos a partir da raíz do projeto. No entanto, eles serão substituídos pelo DP após a submissão. Ou seja, o DP irá correr os testes do professor usando ficheiros diferentes daqueles que os alunos submeterem.

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-2021-projeto-p2

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

Filosofia do uso do DP

Pelas mesmas razões descritas na parte 1, cada grupo apenas pode fazer uma submissão a cada 30 minutos.

Prazo de entrega

A data limite de entrega é o **dia 14 de Junho de 2020 (segunda-feira), pelas 23h00m** (hora de Lisboa, Portugal). Não serão aceites trabalhos após essa data e hora.

As defesas decorrerão nos dias 17 e 18 de Junho.

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.

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. Nota: Para garantir a justiça na avaliação, esta componente só é avaliada para projetos que passem todos os testes (porque um projeto que só implementou metade das queries correrá provavelmente mais rápido que um que implementou todas as queries)	1,5
Testes automáticos	Os alunos devem implementar pelo menos um caso de teste em JUnit para cada query/comando e manter os testes ao `toString()` da sua classe `Song`. Por caso de teste considera-se uma função/método anotada com `@Test`.	1

Participação no piazza	Alunos que participem ativamente no piazza. Ver secção “Esclarecimento de Dúvidas”	1
Criatividade	Pergunta extra (ver secção “Componente de criatividade”)	1
Vídeo	Vídeo exemplificativo da forma como implementaram uma query não obrigatória e da query que inventarem (criatividade). Ver regras na secção “Artefactos a entregar”	1

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
- Projetos que não implementem as queries obrigatórias serão considerados como não entregues e terão nota zero.

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:

– Os projetos devem ser realizados em grupos de **2 alunos**. Excecionalmente poderão ser aceites trabalhos individuais, desde que seja pedida autorização prévia ao Professor das aulas práticas respetivo e desde que exista uma justificação razoável.

– 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.

- Na segunda parte do projeto devem-se manter os grupos definidos para a primeira parte. Não será permitida a entrada de novos membros nos grupos.

- Grupos que não entreguem a primeira parte ou tenham nota zero na mesma (seja por cópia, não compilação ou outra das situações indicadas no enunciado), não podem entregar a segunda parte do projeto, e reprovam automaticamente na componente prática da disciplina (de primeira época).

Anexo I - Exemplos dos ficheiros e de uma sessão interativa

Os ficheiros de exemplo da Parte 1 podem ser usados nesta Parte 2, dado que as sintaxes dos ficheiros são as mesmas.

Serão também publicados em moodle ficheiros com uma grande quantidade de dados, para facilitar os testes de eficiência.

Exemplo de 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
1164
(took 3 ms)

COUNT_DUPLICATE_SONGS_YEAR 2000
20
(took 15 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 | How Long | 1974
Ace Spectrum | I Don't Want to Play Around | 1974
...
(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
Muriel Harding 5
Hoagy Carmichael & His Orchestra 5
...
(took 6 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 5 ms)

CLEANUP
Musicas removidas: 3; Artistas removidos: 2
(took 477 ms)

KTHXBYE
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 DEISI Rockstar!");

Scanner in = new Scanner(System.in);

String line = in.nextLine();

while (line != null && !line.equals("KTHXBYE")) {
   long start = System.currentTimeMillis();
   String result = execute(line);
   long end = System.currentTimeMillis();
   System.out.println(result);
   System.out.println("(took " + (end - start) + " ms)");

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