



Enunciado do projeto prático



Primeira Parte

Objetivos

Este projeto tem como objetivo o desenvolvimento de um programa/aplicação (o “**Rock in DEISI 2023**”) em linguagem Java que seja capaz de receber e responder a pedidos de informações/estatísticas sobre uma base de dados de músicas e artistas (bandas, músicos, etc.).

O programa terá de ler dados dessa base de dados, que será fornecida a partir de ficheiros num formato especificado mais à frente.

Tendo em conta que um dos temas tratados em **AED** é a questão da **eficiência** das estruturas de dados e dos algoritmos, pretende-se que o programa desenvolvido seja não só funcionalmente correto, mas também que seja o mais rápido possível.

Para alcançar estes objetivos, os alunos devem ter em conta tudo o que foram aprendendo em AED. Recomenda-se que façam abordagens exploratórias, fazendo medições dos tempos de execução de cada sub-passo do programa, no sentido de encontrar eventuais *bottlenecks* e que se dediquem a tentar melhorar os tempos de execução dos mesmos.

“Make it work, Make it right, Make it fast.”

Organização do trabalho

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

Este é o enunciado da **Primeira Parte**.

O enunciado da Segunda Parte será publicado em *moodle* em data posterior.

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

Os trabalhos devem ser realizados em grupos de 2 alunos.

Em situações excepcionais poderão ser aceites grupos de um aluno - quem o pretender fazer deve efetuar um pedido (justificado) aos docentes da cadeira através de e-mail.

Recomendamos que formem grupos com alunos de “nível”¹ equivalente. Pela nossa experiência, quando há uma discrepância grande (ex: um dos alunos teve 18 a Fundamentos de Programação e o outro reprovou), o processo de aprendizagem para o aluno com mais dificuldades fica fortemente comprometido, culminando frequentemente em reprovação na defesa. É preferível que um aluno com dificuldades forme grupo com outro aluno na mesma situação, pois vão trabalhar juntos para ultrapassar as dificuldades, potenciando o processo de aprendizagem.

Objetivos da Primeira Parte

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

- Definição e implementação de classes Java para representação das entidades da realidade/domínio do problema;
- Implementação da Leitura dos Ficheiros que são o *input* do programa;
- Criação (instanciação) de objectos em memória, considerando os ficheiros lidos.

Importante: Na primeira parte não será ainda avaliada a eficiência dos algoritmos. Os ficheiros utilizados nos testes serão pequenos, pelo que eventuais ineficiências não terão impacto significativo. O foco deverá ser portanto na execução correta do programa - garantir que funciona corretamente para quaisquer ficheiros que sejam usados como input.

¹ Podem usar as notas de Fundamentos de Programação (nomeadamente na parte prática) e do mini-teste 1 para aferirem o nível de cada um.

Realidade / Domínio do Problema

Tal como o nome do projeto indica, o domínio deste problema está centrado na música.

No mundo da música, os principais elementos são os artistas que podem atuar a solo ou como uma banda. Todos os artistas têm um nome mas, como podem haver nomes repetidos, será melhor identificar cada um de forma única. E o que fazem esses artistas? Interpretam temas musicais com as mais diversas características que vão possivelmente influenciar a sua popularidade. Será que têm a capacidade de fazer as pessoas dançar? Ou são simplesmente mais ou menos animadas (sem relação com dança)?

Uma coisa é certa: há temas musicais cujo nome é conhecido de toda a gente mas, tal como os artistas, podem haver nomes repetidos. Por isso, também é necessário associar a cada tema um identificador único.

Alguns anos foram particularmente importantes na história da música. Por exemplo, o movimento grunge teve o seu auge em 1991, com o lançamento do álbum “Nevermind” dos Nirvana, o álbum “Ten” dos Pearl Jam, “Badmotorfinger” dos Soundgarden e “Dirt” dos Alice in Chains. Uma coisa que todos estes álbuns tinham em comum era terem temas com um elevado número de decibéis (não muito saudável para os nossos ouvidos) e algumas letras serem explícitas (isto é, usarem palavras ou expressões que podem ser consideradas ofensivas ou inadequadas para alguns ouvintes).

Uma das coisas mais interessantes que acontece no mundo da música é quando alguns artistas se juntam para interpretar uma música. Por exemplo, a música “Lady Marmalade” que aparece na banda sonora de Moulin Rouge foi interpretada por Christina Aguilera, Lil’ Kim, Mya e Pink, 4 artistas com uma carreira própria que se juntaram especificamente para gravar esta música.

Dados de entrada (*Input*) do programa

O *input* do programa consiste em **três** ficheiros de texto:

Nome do ficheiro	Conteúdo
songs.txt	Este ficheiro descreve os temas musicais.
song_artists.txt	Este ficheiro descreve quais os artistas que participam em cada tema musical. (Nota: Algumas músicas poderão não ter artistas associados)
song_details.txt	Este ficheiro descreve diversos detalhes sobre os temas musicais como a sua duração, popularidade, etc. (Nota: Poderão faltar alguns detalhes para algumas músicas)

Não são conhecidas, à priori, as dimensões (número de linhas) destes ficheiros.
Para implementarem a leitura de ficheiros, recomendamos a visualização prévia do vídeo “Leitura de ficheiros e exceptions” colocado no Moodle.

De seguida são apresentadas as sintaxes de cada um dos ficheiros.

Ficheiro songs.txt

Cada linha deste ficheiro descreve um **Tema Musical** e segue a seguinte sintaxe:

```
<ID Tema Musical> @ <Nome> @ <Ano lançamento>
```

Podem ver exemplos destas linhas no anexo I (final do documento)

Tratamento de erros

Como se pode ver, este ficheiro deverá ter 3 componentes em cada linha.

No entanto, pode acontecer que existam ficheiros com dados a mais, ou dados a menos. **As linhas que tenham dados a mais ou a menos devem ser ignoradas.**

No mundo perfeito, não haveriam duas linhas com o mesmo ID. Mas o mundo real não é perfeito... Tenham isso em atenção e ignorem essas linhas (apenas aceitam a primeira linha com cada ID).

Algumas músicas poderão não ter detalhes associados (ver secção “Ficheiro song_details.txt”). Nesse caso, devem ser ignoradas. No entanto, não devem ser contabilizadas como linhas com erro, para a informação estatística de leitura do ficheiro, uma vez que o ficheiro “songs.txt” só por si não tem nenhum erro.

Ficheiro song_artists.txt

Cada linha deste ficheiro segue uma das seguintes sintaxes consoante a música está associada a apenas um artista ou vários artistas:

```
<ID Tema Musical> @ [ '<Nome Artista>' ]  
<ID Tema Musical> @ "[ '<Nome Artista>', '<Nome Artista>', ... ]"
```

Onde:

- <ID Tema Musical> é uma String;
- <Nome Artista> é uma String;

Como se pode ver, os nomes dos artistas estão sempre entre plicas. Caso sejam vários estão separados por ` , ' (vírgula). Todo o conjunto de artistas (seja um ou vários) está sempre entre parênteses rectos mas caso sejam vários, têm que estar entre aspas.

Em certas linhas podem haver espaços entre os símbolos que devem ser ignorados.

Por exemplo (espaços, representados a verde, devem ser ignorados):

```
1434545625 @ [ [ 'The National' ] ]
```

Notas:

O mesmo artista pode estar associado a vários Temas Musicais.

Algumas músicas poderão não estar associadas a nenhum artista. Nesse caso, devem ser ignoradas. No entanto, não devem ser contabilizadas como linhas com erro, para a informação estatística de leitura do ficheiro, uma vez que o ficheiro “songs.txt” só por si não tem nenhum erro.

Tratamento de erros

Como se pode ver, este ficheiro deverá ter 2 componentes em cada linha. No entanto, pode acontecer que existam ficheiros com dados a mais, ou dados a menos. **As linhas que tenham dados a mais ou a menos devem ser ignoradas.**

Caso apareçam várias linhas associadas à mesma música com artistas diferentes, essas linhas deve ser consideradas válidas. Todos os artistas (das várias linhas) estarão associados a essa música.

Ficheiro song_details.txt

Cada linha do ficheiro segue a seguinte sintaxe (tudo na mesma linha):

```
<ID Tema Musical> @ <Duração> @ <Letra explícita> @  
<Popularidade> @ <Dançabilidade> @ <Vivacidade> @ <Volume médio>
```

Podem ver exemplos destas linhas e respetivo formato no anexo I (final do documento). Se tiverem dúvidas perguntem no discord.

Tratamento de erros

Como se pode ver, este ficheiro deverá ter 7 componentes em cada linha. No entanto, pode acontecer que existam ficheiros com dados a mais, ou dados a menos. **As linhas que tenham dados a mais ou a menos devem ser ignoradas.**

Dados de saída (Output) do programa

Nesta primeira parte do trabalho não há necessidade de fazer qualquer *output*.

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**.

As restrições são as seguintes:

- Além da classe `Main`, têm que criar entre 2 a 5 classes (inclusivé).
- Devem criar classes suficientes para representar as entidades descritas no domínio do problema
- Todas as classes criadas devem ter pelo menos um construtor não vazio e a função `toString()`. Mais abaixo será descrito o formato com que deve ser mostrada a informação de certas entidades, devem ter isso em conta quando criarem a função `toString()`. Nos casos em que nada seja dito sobre o formato, podem inventar o formato que vos fôr mais conveniente.
- Deverão criar um enumerado chamado `TipoEntidade` que pode ter os valores `TEMA`, `ARTISTA` e `INPUT_INVALIDO`.
 - No Anexo II deste enunciado encontram-se exemplos da declaração e uso de tipos enumerados (vulgo *enums*) em Java.
- A classe `Main` deve incluir obrigatoriamente as seguintes funções:

```
public static boolean loadFiles(File folder)
public static ArrayList getObjects(TipoEntidade tipo)
```

- A função **`loadFiles(File folder)`**, ao ser invocada, deverá:
 - ler os três ficheiros de input a partir da pasta passada como parâmetro e criar objetos das classes que criaram previamente com os dados existentes nos ficheiros;
 - guardar esses objetos em estruturas de dados apropriadas (arrays, arraylists, dentro ou fora dos objetos);
 - deve retornar `true` caso tenha conseguido ler os ficheiros ou `false` caso contrário;
 - Dica: Para criar um objeto `File` que representa a raiz do projeto, podem usar `new File(".")`
 - Dica: Para referenciar um ficheiro dentro de uma pasta, podem usar `new File(folder, "nomeDoFicheiro.txt")`

- A função `getObjects(TipoEntidade tipo)` deverá retornar um `ArrayList` com todos os objetos (válidos) do tipo indicado, que existem na base de dados carregada a partir dos ficheiros.
 - A ordem pela qual as entidades aparecem nos respetivos ficheiros deve ser preservada.
 - Notem que os testes vão validar os objetos retornados através da respetiva função `toString()`.
- Para objetos contendo informação de músicas (quando o tipo é `TEMA`), o formato pretendido quando estes são convertidos para `String` assume 3 opções de dificuldade crescente:
 - Quando a música é de um ano inferior a 1995:

ID Título Ano

Exemplo:

1oYYd2gnWZYrt89EBXdFiO Message In A Bottle 1979

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:

"1oYYd2gnWZYrt89EBXdFiO|Message In A Bottle|1979"

não será considerado correto, pois não tem espaçamento certo.

- Quando a música é de um ano entre 1995 e 2000:

ID Título Ano Duração Popularidade
--

em que *Duração* deve ser representada nesta forma: minutos:segundos. Por exemplo: 3:41 representa 3 minutos e 41 segundos.

- Quando a música é de um ano igual ou superior a 2000:

ID Título Ano Duração Popularidade NumArtistas

em que *NumArtistas* é um inteiro com o número de artistas que colaboraram na interpretação dessa música

- Para objetos contendo informação de músicos (quando o tipo é `ARTISTA`), o formato pretendido quando estes são convertidos para String assume 2 opções de dificuldade crescente:

- Quando o nome do artista começa pela letra A, B, C ou D:

```
Artista: [Nome do artista]
```

em que o nome do artista aparece dentro dos parêntesis retos em aspas e sem espaços à volta

- Quando o nome do artista começa por outra letra que não o caso anterior:

```
Artista: [Nome do artista] | NumMusicas
```

em que NumMusicas é um inteiro com o número de músicas em que este artista colaborou

- Quando o tipo é `INPUT_INVALIDO`, a função deve retornar 3 objetos da mesma classe, com informação estatística da leitura de cada um dos 3 ficheiros. Cada objeto deverá estar associado às estatísticas de um ficheiro: o primeiro objeto deve estar associado ao ficheiro `songs.txt`, o segundo ao ficheiro `song_details.txt` e o terceiro ao ficheiro `song_artists.txt`. Quando esses objetos são convertidos para String, pretende-se a seguinte informação:

```
NomeFicheiro | LinhasOK | LinhasNOK | PrimeiraLinhaNOK
```

Em que:

- LinhasOK - Número de linhas sem erros
- LinhasNOK - Número de linhas com erros que, ou seja, que foram ignoradas
- PrimeiraLinhaNOK - Número da primeira linha com erro (nota: a primeira linha do ficheiro corresponde à linha nº 1). Caso não existam linhas com erro, deve ficar com o valor -1.

Exemplo:

```
songs.txt | 341 | 7 | 29
```

- Os testes executarão sempre a função `loadFiles(...)` antes de executarem a função `getObjects(...)`.

- Nesta parte, não é permitida a utilização da classe Set ou Map ou qualquer uma das suas variantes (HashSet, TreeSet, HashMap, LinkedHashMap, TreeMap, ConcurrentHashMap, etc.). Também não é permitida a utilização de streams.
- Todas as classes do projeto devem pertencer ao seguinte *package*:

```
pt.ulusofona.aed.rockindeisi2023
```

Dica:

Os testes automáticos dos professores podem chamar a função `loadFiles(...)` mais do que uma vez. Os alunos devem garantir que nunca ocorre duplicação de informação no `ArrayList` devolvido pela função `getObjects(...)` (p.e. fazendo um *reset* das estruturas de dados relevantes).

Testes Unitários Automáticos

Nesta componente, os alunos devem implementar **casos de teste unitários automáticos** para garantir a qualidade do seu projecto. Por “caso de teste” entende-se a definição de uma função (anotada com `@Test`) que verifique se, para certo “*input*” é obtido o “*output*” / *resultado esperado*.

Em particular, espera-se que os alunos criem pelo menos **6 funções de teste** para cada um destes cenários:

- Conversão para String de objetos contendo informação de músicas anteriores a 1995
- Conversão para String de objetos contendo informação de músicas anteriores a 2000
- Conversão para String de objetos contendo informação de músicas superiores a 2000
- Conversão para String de objetos contendo informação de artistas
- Leitura dos 3 ficheiros (sem erros) contendo pelo menos 2 músicas e 2 artistas
- Leitura dos 3 ficheiros com erros (linhas com elementos em falta)

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

Os casos de teste devem ser implementados usando **JUnit 5**. Recomenda-se a visualização do vídeo “Como desenvolver testes em JUnit” colocado no Moodle. O vídeo refere-se ao JUnit 4

mas será disponibilizado em breve um novo vídeo atualizado para o JUnit 5. 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
|-----+ rockindeisi2023
|----- Main.java
|----- ... (outros ficheiros java do projecto)
|----- TestXXX.java
+ test-files
|--- songs.txt
|--- ...
```

Notem que os testes unitários irão ler os ficheiros de uma pasta diferente da usada pelo `main()`. O `main()` lê da raiz enquanto os testes unitários lêem da pasta “test-files”. Para diferenciar, deverão fazer uso do parâmetro `folder` que é passado à função `loadFiles`.

Apresenta-se um exemplo de cada:

```
public static void main(String args[]) {

    boolean result = loadFiles(new File("."));

    if (result) {
        ...
    }
}
```

```
}
```

```
@Test
public void testBanana {

    boolean result = loadFiles(new File("test-files"));

    if (result) {
        ...
    }
}
```

Utilização do chatgpt

Tal como tem sido referido nas fichas práticas semanais, devem evitar recorrer ao chatgpt constantemente. Comecem por tentar implementar sozinhos as funções do projeto antes de recorrerem ao chatgpt. Lembrem-se que ele pode ser usado para vos apoiar na construção do programa e nunca para vos substituir completamente.

No entanto, há uma parte específica do projeto em que incentivamos que usem mesmo o chatgpt para vos ajudar - no processamento das linhas do ficheiro `song_artists.txt`, pois têm várias particularidades que complicam a sua implementação (de forma a contemplar todos os casos).

Mas devem usar o vosso espírito crítico nesse processo. Devem obter do chatgpt duas possíveis implementações para esse processamento e documentá-las num ficheiro que deverão incluir no vosso projeto. **O ficheiro deve ter o nome `README.txt`** e deve estar na raiz do projeto, junto ao ficheiro `AUTHORS.txt`.

Dentro desse ficheiro devem colocar:

- O “diálogo” que tiveram com o chatgpt para obter as 2 implementações.
- Qual das implementações escolheram e porquê

Exemplo abaixo (notem que colocámos “(...)” para que o texto coubesse neste enunciado mas no vosso ficheiro devem colocar a resposta completa):

Humano: implementa uma função em Java para determinar o maior número num array de inteiros

Chatgpt: Claro! Aqui está uma possível implementação em Java para encontrar o maior número em um array de inteiros:

```
public static int encontraMaior(int[] numeros) {
    int maior = numeros[0]; // começa com o primeiro número do array
    for (int i = 1; i < numeros.length; i++) {
        if (numeros[i] > maior) {
            maior = numeros[i]; // atualiza o maior número
        }
    }
    return maior;
}
```

Esta função recebe um array de inteiros numeros e retorna o maior número encontrado. A função começa definindo o primeiro número do array como o maior. Em seguida, percorre o restante do array, (...)

Humano: Implementa uma versão alternativa dessa função que não inicialize a variável "maior" com o primeiro número do array

Chatgpt: Claro! Aqui está uma possível implementação alternativa em Java para encontrar o maior número em um array de inteiros sem inicializar a variável maior com o primeiro número do array:

```
public static int encontraMaior(int[] numeros) {
    int maior = Integer.MIN_VALUE; // começa com o menor valor possível para um inteiro
    for (int i = 0; i < numeros.length; i++) {
        if (numeros[i] > maior) {
            maior = numeros[i]; // atualiza o maior número
        }
    }
    return maior;
}
```

Esta função começa definindo a variável maior com o menor valor possível para um inteiro usando a constante Integer.MIN_VALUE. Em seguida, percorre o array, (...)

CONCLUSÃO DO GRUPO: Escolhemos a primeira opção pois evita uma comparação extra no início do ciclo (o ciclo pode começar na posição 1 em vez da posição 0).

Entrega e Avaliação

O que é preciso entregar?

Nesta primeira parte do projeto, os alunos têm de entregar:

- Todo o código necessário para compilar e executar o projeto;
- 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 as interações com o chatgpt (ver secção “Utilização do chatgpt”)

A entrega deve ser feita usando um ficheiro comprimido (formato .zip).

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   (contém interações com o chatgpt)

... (outros ficheiros)
+ src
|---+ pt
|-----+ ulusofona
|-----+ aed
|-----+ rockindeisi2023
|----- Main.java
|----- ... (outros ficheiros java do projecto)
|----- TestXXX.java
|----- ... (outras classes de testes)
```

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

Nota importante: Os ficheiros de *input* devem ser lidos a partir da raiz do projeto.

Como entregar - Drop Project

Os alunos terão acesso a uma página no **Drop Project** a partir da qual poderão submeter o seu projeto e obter um relatório sobre os testes aplicados ao mesmo.

O DP da primeira parte do projeto estará disponível no URL seguinte:

<https://deisi.ulusofona.pt/drop-project/upload/aed-2223-projeto-p1>

Nota importante: O programa submetido pelos alunos tem de compilar e executar no contexto do Drop Project.

Filosofia do uso do DP

O objetivo do DP não é servir de guião àquilo que os alunos têm que implementar. Os alunos devem implementar o projeto de forma autónoma tendo apenas em conta o enunciado e usar o DP apenas para validar que estão no bom caminho. Nas empresas nas quais um dia irão trabalhar não vão ter o DP para vos ajudar.

Desta forma, as submissões no DP serão limitadas temporalmente: cada grupo apenas pode fazer uma submissão a cada 20 minutos (isto é, sempre que fazem uma submissão, têm que esperar 20 minutos até que possam fazer outra submissão).

Prazo de entrega

A data limite de entrega é o **dia 23 de Abril de 2023, pelas 22h00m** (hora de Lisboa, Portugal). Não serão aceites trabalhos após essa data e hora.

Notem que o Moodle entra em manutenção todas as noites, mais ou menos entre as 2 e a 6 da manhã, pelo que poderão haver períodos de indisponibilidade nesse horário.

Notem também que a disponibilidade dos professores para responder a dúvidas durante o fim de semana poderá ser mais reduzida, antecipem ao máximo as vossas dúvidas para evitarem ficar sem resposta.

Avaliação

A avaliação do projeto será dividida pelas 2 partes:

- Nas duas partes existirão baterias de testes automáticos implementadas no sistema **Drop Project** ([DP]) que irão avaliar o projeto do ponto de vista funcional.
- Existirá uma nota em cada parte do projeto.
- Na primeira parte do projeto existe a nota mínima de 8 (oito) valores.
 - Quem não alcançar essa nota mínima ficará excluído da avaliação de primeira época.
- Após a entrega final, será atribuída ao projeto uma nota final quantitativa, que será calculada considerando a seguinte fórmula:
 - $0.25 * \text{Nota_Parte_1} + 0.75 * \text{Nota_Parte_2}$
- No final, existirá uma defesa presencial do projeto.
 - Mais informações na secção “**Outras informações relevantes**” deste enunciado.

Cotações da primeira parte

Item	Descrição	Cotação
Componente funcional	Testes automáticos do DP.	14
Avaliação do Professor	Os Professores irão realizar testes extra das funcionalidades a implementar.	3
Qualidade do código (checkstyle)	Alguns indicadores da qualidade de código como variáveis com nomes auto-explicativos ou funções que não sejam demasiado grandes	1
Testes automáticos	Ver secção “Testes unitários automáticos”	1.5
Utilização documentada do chatgpt	Ver secção “Utilização do chatgpt”. Será avaliada a inclusão do ficheiro README.txt documentando as interações com o mesmo e a respetiva conclusão.	0.5

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.

Esclarecimento de dúvidas

Todas as dúvidas relativas ao projeto (seja de interpretação do enunciado, seja da forma como se implementa) devem ser colocadas na ferramenta **discord**, cujo endereço foi já divulgado via Moodle.

Outras informações relevantes

- Os grupos de alunos definidos para a primeira parte do projecto devem ser mantidos para a segunda parte do projecto. Não serão permitidas entradas de novos alunos nos grupos entre as duas partes do projecto.
- A qualquer altura do semestre os Professores podem convocar o aluno para uma prova oral sobre o seu projecto. Esta prova tem como objectivo determinar a participação do aluno no projecto, e pode resultar na atribuição da nota zero (0) ao aluno.
- Existirá uma defesa presencial e individual do projeto, após entrega da parte 2. 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.

- O ficheiro .zip entregue deve seguir escrupulosamente as regras de estrutura indicadas neste enunciado. Ficheiros que não respeitem essa estrutura terão nota zero.
- 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).
- Não deve ser implementado qualquer menu (ou interface gráfica) para interação manual com o utilizador.
- É possível que sejam feitas alterações a este enunciado, durante o tempo de desenvolvimento do projeto. Por esta razão, os alunos devem estar atentos ao Moodle de AED.
- Eventuais situações omissas e/ou imprevistas serão analisadas caso a caso pelos docentes da cadeia.

Anexo I - Exemplos dos ficheiros de input

Neste anexo é apresentado um exemplo de cada um dos três ficheiros de *input*.

Este exemplo deve ser usado pelos alunos para testarem o **comportamento** do seu programa, antes de o submeterem para avaliação.

Ficheiros

songs.txt

```
17ZnveSDBpG9QtL7zLJNPY @ Only For You @ 2012
0tStWvUMHODuLN4TIaSGab @ Loops & Tings - Radio Edit @ 2013
37zoJIMxB98H2fbidrxuO @ Downshift - Original Mix
1sNctqVr9zdS7ilRZNjIgY @ Crazy Kids @ 2012
1SlMYQmLe0yNEvBifaPTAW @ Ich will @ 2001
1KLyMQwUeIPJaXn7wgdeNW @ Heirate mich @ 1995
1l4CEjapdCSUsazfBMjR4l @ The Lady Is a Tramp @ 1948
6r8klvznHrzlEKYxL4dZEe @ La Isla Bonita @ 1986
1fEepJX6lZ8fqhDRsNxkQp @ Overture @ 1956
0nxvFG50rGXkiGQqOO2Mhr @ Be Alright @ 2012
```

Nota: a terceira linha tem apenas 2 componentes, em vez dos 3 esperados. Por esta razão, essa linha tem de ser ignorada.

song_artists.txt

```
7dD5DEzjhofItSG7QwVoY @ ['Frank Sinatra']
5TnAnWRXDM6awfvAveK91N @ ['Mountain John']
5tNhreO6L8kfvQSQpjuGUg @ ['Rammstein']
4Cna7QxOOTNnylVHLtShCi @ ['Madonna']
2Sylr4fGfq4Lslfc0gHkbbk @ "['La Sonora Matancera', 'Carlos Argentino']"
3uXp4hZmCnEmVjffupKCiT @ ['Supertramp']
1SlMYQmLe0yNEvBifaPTAW @ ['Rammstein']
1KLymQwUeIPJaXn7wgdeNW @ ['Rammstein']
1l4CEjapdCSUsazfBMjR4l @ ['Rosemary Clooney']
6r8klvznHrzlEKYxL4dZEe @ ['Madonna']
1fEepJX6lz8fqhDRsNxkQp @ "['Antônio Carlos Jobim', 'Vinícius de Moraes']"
```

song_details.txt

```
1SlMYQmLe0yNEvBifaPTAW @ 217107 @ 0 @ 57 @ 0.648 @ 0.0879 @ -4.254
1KLymQwUeIPJaXn7wgdeNW @ 284667 @ 0 @ 45 @ 0.649 @ 0.23 @ -8.906
1l4CEjapdCSUsazfBMjR4l @ 125267 @ 0 @ 19 @ 0.691 @ 0.222 @ -12.895999999999999
1fEepJX6lz8fqhDRsNxkQp @ 410933 @ 0 @ 16 @ 0.221 @ 0.174 @ -10.066
6r8klvznHrzlEKYxL4dZEe @ 242320 @ 0 @ 70 @ 0.725 @ 0.0548 @ -10.706
0xJ6Xr620TwVr6kqDSk5tn @ 347951 @ 0 @ 51 @ 0.397 @ 0.127 @ -6.678
2GKyRxxh2DAC0sSVddqUE4l @ 178077 @ 0 @ 4 @ 0.292 @ 0.0993 @ -8.497
1lxVGQgbt0g7mMkmevl9Bm @ 205613 @ 0 @ 34 @ 0.608 @ 0.115 @ -10.265999999999998
0GmLrYUBXDC5vti77zBZfJ @ 167680 @ 0 @ 36 @ 0.5770000000000001 @ 0.226 @ -9.825
```

Anexo II - Exemplos de uso de Enumerados

Os tipos enumerados são uma ferramenta poderosa para criar um conjunto de constantes relacionadas entre si. Em Java, os tipos enumerados fornecem uma forma segura e fácil de trabalhar com essas constantes. Os tipos enumerados permitem criar variáveis que possam assumir apenas um conjunto de valores predefinidos. Estes valores são chamados de membros da enumeração.

Um exemplo de uso de tipos enumerados em Java é criar uma enumeração chamada "EstadoCivil" que tenha os membros Solteiro, Casado, Divorciado e Viúvo. O código para criar a classe enumerada seria o seguinte:

```
public enum EstadoCivil {  
    SOLTEIRO, CASADO, DIVORCIADO, VIUVO;  
}
```

Outro exemplo de uso de tipos enumerados em Java é criar uma enumeração chamada "Tamanho" que tenha os membros Pequeno, Médio e Grande. O código para criar a classe enumerada seria o seguinte:

```
public enum Tamanho {  
    PEQUENO, MEDIO, GRANDE;  
}
```

Tal como as classes, os *enums* definem novos “tipos de dados”, o que significa que podemos criar variáveis com esses tipos.

Exemplo:

```
public static void main(String[] args) {  
    EstadoCivil estadoCivil = EstadoCivil.CASADO;  
    // ...  
    if (estadoCivil == EstadoCivil.CASADO) {  
        System.out.println("Você é casado!");  
    }  
}
```