Exame de "Introdução à Programação" 2008-09
19 de Fevereiro de 2008

Duração: 2:30H

Instruções muito importantes:

- Responda a cada grupo em folhas **separadas**.
- Identifique **todas** as folhas com o seu número e nome.
- Antes de começar a resolver, leia o enunciado do **princípio até ao fim**.
- **Pode** usar caneta ou lápis.
- Não é permitido consultar quaisquer elementos para além deste enunciado.
- Qualquer tentativa de fraude comprovada acarretará a reprovação na disciplina.
- Não é permitido sair da sala antes que o exame termine.

I.

Pretende-se programar em Java uma classe **`Music`** cujos objectos representam músicas. Cada música é caracterizada pela seguinte informação: nome da música, nome do artista ou banda, duração (em segundos), formato do ficheiro (mp3, wav, aiff) e classificação (de 1 a 5).

O construtor da classe **`Music`** é definido da seguinte forma:

- **`public`** `Music(String musicName, String artist, int duration, String fileFormat, int rating)`: Constrói um objecto **`Music`** com o nome `musicName`, artista `artist`, duração `duration`, formato `fileFormat` e classificação `rating`;

A interface pública da classe **`Music`** é definida da seguinte forma:

- **`public`** `String getName()`: Devolve o nome da música;
- **`public`** `String getArtist()`: Devolve o artista da música;
- **`public int`** `getDuration()`: Devolve a duração da música;
- **`public`** `String getFormat()`: Devolve o formato da música;
- **`public int`** `getRating()`: Devolve a classificação da música;
- **`public boolean`** `equals(`**`Music`**`m)`: Indica se duas músicas são iguais. Considere que duas músicas são iguais quando têm o mesmo nome, o mesmo artista e a mesma duração;
- **`public Music`** `convert(String fileFormat)`: Devolve um novo objecto **`Music`** igual em tudo ao próprio, excepto no formato, que deve passar a ser o indicado por `fileFormat`. O objecto que executa a operação `convert` não deve ser alterado.

Programe, em Java, a classe **`Music`**.

II.

Para construir um gestor de playlists é preciso definir uma classe **`PlayList`** que representa um conjunto de músicas. Cada música é representada por um objecto da classe **`Music`**. É importante garantir que uma música não aparece duplicada na playlist, para esse efeito considera-se que não existem duas músicas iguais na playlist, onde "iguais" é verificado com o método `equals` da classe **`Music`**. Os dois construtores da classe **`PlayList`** são os seguintes:

- **`public PlayList`**`(String name, int n)`: Constrói uma nova **`PlayList`**, com o nome `name`. O parâmetro `n` indica o número máximo de músicas que a playlist pode guardar.
- **`public PlayList`**`(String name)`: Constrói uma nova **`PlayList`**, com o nome `name`. Neste caso, a playlist deve poder conter até 256 músicas.

A interface da classe **`PlayList`** deve conter os seguintes métodos:

- **`public`** `String getName()`**:** Devolve o nome da playlist;
- **`public int`** `getSize()`**:** Devolve o número de músicas da playlist;
- **`public int`** `getDuration()`**:** Devolve o tempo total de reprodução de todas as músicas na playlist.
- **`public boolean`** `contains(`**`Music`** `m)` : Devolve **`true`** se a música `m` pertence à playlist, e **`false`** caso contrário;
- **`public boolean`** `addMusic(`**`Music`** `m)` : Adiciona a música `m` à playlist, se possível, devolvendo **`true`**. Caso a mesma música já exista na playlist, ou a playlist não possa admitir mais músicas, esta operação não faz nada e devolve **`false`**.
- **`public boolean`** `deleteMusic(`**`Music`** `m)` : Se a música `m` pertence à playlist, remove-a e devolve **`true`**, caso contrário, não faz nada, mas devolve **`false`**.
- **`public int`** `numMusics(String artist)`**:** Devolve o número de músicas na playlist que são do artista `artist`;

a) Programe a classe **`PlayList`** em Java.

b) Escreva um programa principal em Java que leia, do ficheiro de texto "tunes.txt", um conjunto de músicas e as insira numa playlist (representada por um objecto **`PlayList`**).

Depois de ler toda a informação, o programa deve pedir ao utilizador o nome de um artista, e responder o número de músicas na playlist que são do artista indicado.

O ficheiro de texto "tunes.txt" tem o seguinte formato: na primeira linha aparece um número inteiro positivo N que indica o total de músicas. Nas seguintes linhas aparecem as informações das várias (N) músicas. Para cada música aparece o seu nome, o artista, a duração, o formato do ficheiro, e a classificação, todos estes elementos em linhas diferentes.

III.

Considere que pretende desenvolver uma aplicação de gestão de músicas **yTunes**. Neste exercício, **não terá que programar**, mas apenas que definir a lista de operações assim como as variáveis e constantes que achar necessárias para implementar a memória (ou estado) de cada objecto/classe. Pode assumir como já existentes as classes `Music` e `PlayList` consideradas nos pontos anteriores do exame.

O **yTunes** deverá:

- Manter, numa biblioteca global, informação associada a cada música armazenada.
- Inserir e remover músicas da biblioteca global. Será esta a única maneira de alterar as músicas que um **yTunes** contém.
- Criar e apagar playlists. Quando uma playlist é criada, ela está vazia. Cada playlist deve ter um nome. Deve ser possível criar até 64 playlists diferentes. Quando se remove uma playlist, as músicas que esta continha continuarão a estar na biblioteca global.
- Registar numa playlist uma música existente na biblioteca global (mantendo essa música na biblioteca global).
- Remover uma música existente numa playlist (sem a retirar da biblioteca global).

Nota: se se remover uma música da biblioteca global, ela deverá ser removida também de **todas** as playlists que a contiverem. Descreva (**sem a implementar!**) a classe de que necessitaria para implementar o **yTunes**. Mais concretamente:

a) Defina quais são as operações (e os construtores) necessários, descrevendo a sua funcionalidade através de um comentário. Para cada operação indique, claramente, os parâmetros necessários e o que a operação devolve (se é que devolve alguma coisa). Note que **não tem** que programar as operações.

b) Defina quais são as variáveis da classe. Para cada uma indique o seu tipo, e escreva uma pequena explicação sobre o que ela representa. No caso de necessitar de constantes, defina ainda os valores que lhes serão atribuídos.

IV.

Nota: As questões I, II e III valem 80% do exame.

a) Considere de novo a classe Java **`Music`** da secção I. Acrescente à classe **`Music`** a seguinte operação:

- **`public`** `String getRatingPict()` : devolve uma string com um número de estrelas (carácter "*") igual à classificação da música. Por exemplo, se a classificação for 3, devolve "***".

b) Considere de novo a classe Java **`Playlist`** da secção II. Acrescente à classe **`Playlist`** as seguintes operações, que vão servir para percorrer todas as músicas numa playlist:

- **`public void`** `initIteration()` : acciona o início de um percurso sobre todas as músicas da playlist.
- **`public boolean`** `hasNextMusic()`: indica se existe ainda alguma música no percurso corrente;
- **`public Music`** `nextMusic()`: devolve a música corrente no percurso, e avança para a música seguinte. Este método só deve ser executado, caso a chamada a `hasNextMusic()` devolva o resultado `true`.

c) Usando os métodos acima, escreva um método que, dada uma playlist, escreva num ficheiro chamado "screen.txt" uma lista com os nomes das suas músicas e respectivas classificações, indicadas pictoricamente através de estrelas. Por exemplo,

Scar Tissue *****

Aguas de Março ****

Hotel California **

RoadHouse Blues ***

Rock Lobster *

Nota: se em vez de escrever no ficheiro escrever no ecrã, perde metade da cotação.

d) Acrescente à classe **`Playlist`** um método (invente um nome apropriado) que deve devolver o nome do artista com mais músicas na playlist. Em caso de empate, deve devolver o artista que tem a música com a maior duração. Em caso de novo empate deve devolver um qualquer de entre os empatados.