Programação Centrada em Objectos (PCO)
Programação Por Objectos (PPO)

Projecto de Programação I (até 30 de Outubro de 2015)

1 Introdução

O projecto envolve a representação de *melodias*, em que uma melodia é representada por uma sequência de *notas* musicais. Para a realização do trabalho não são necessários quaisquer conhecimentos preliminares sobre música, mas apenas ter presente a representação de notas segundo uma convenção simples.
O seu trabalho, **a entregar até 30 de Outubro**, consistirá na implementação das seguintes classes:

- `Note` para representar notas musicais;
- `Melody` para representar melodias;
- e `MelodyIO` para operações de leitura/gravação de melodias de/para ficheiros de texto.

É fornecido um projecto Eclipse base contendo o esqueleto das classes em causa (em termos de assinaturas de métodos a implementar), exemplos de melodias em ficheiros de texto (pasta `examples`), e as seguintes classes auxiliares (**que não deverá modificar**):

- `NotePlayer`: uma classe que permite reproduzir notas com um som de piano;
- `Program`: um programa de teste com interface gráfico para reproduzir e manipular melodias;
- as enumerações `Pitch` e `Acc`, auxiliares à representação de notas (ver Secção 3).

Leia também atentamente as orientações para avaliação, a descrição dos requisitos para as classes nas secções, bem como sugestões para iniciar o trabalho na parte final. Use também o **projecto Eclipse "demo"** contendo uma aplicação que permite ilustrar toda a funcionalidade.

2 Critérios de avaliação

1. O seu código deverá funcionar correctamente e usar classes/objectos de forma adequada.
2. O seu código deverá ser organizado, documentado e fácil de ler, i.e., incluir comentários Javadoc bem formatados e código com um estilo de programação "limpo" e coerente, ex. em termos de indentação, uso adequado de nomes para variáveis, campos, etc. Veja informação no guião *Uma questão de estilo – Elementos de estilo Java*, V. Vasconcelos, DI/FCUL.
3. O seu código deverá incluir anotações `@requires` para anotar pré-condições de métodos quando adequado.
4. O código de todos os grupos será analisado por uma ferramenta de detecção automática de similaridade entre trabalhos. Em caso de cópia, os trabalhos dos grupos envolvidos serão anulados.

3 A classe `Note`

Considera-se que uma nota, no espírito da notação SPN, é definida por:

- uma duração, expressa em segundos com um valor de tipo `double`;
- um tipo ("pitch"), representado pela enumeração `Pitch` (já fornecida), com valores possíveis `A` a `G` e ainda o valor especial `S` para denotar silêncio – a correspondência com a designação de notas em português é dada por: C/Dó, D/Ré, E/Mi, F/Fá, G/Sol, A/Lá, e B/Si;
- um "acidente" representado pela enumeração `ACC`, com valores `NATURAL`, `FLAT` (em português: bemol) e `SHARP` (sustenido) – o valor é irrelevante se a nota representar silêncio (`Pitch.S`);
- a "oitava" da nota, que que determina a escala da frequência da nota, expressa por um valor de tipo `int` de 0 a 9 – – o valor é irrelevante se a nota representar silêncio (`Pitch.S`);

Objectos de tipo `Note` deverão representar estes atributos de forma imutável, com campos de instância privados. A classe deverá conter os métodos descritos na tabela a seguir.

Método(s)	**Descrição**
`Note(double duration, Pitch pitch, int octave, Acc acc)`	Constructor para uma nota arbitrária.
`Note(double duration)`	Constructor para silêncio (`Pitch.S`) com a duração especificada.
`double getDuration()`	Devolve duração da nota.
`Pitch getPitch()`	Devolve tipo ("pitch") da nota.
`int getOctave()`	Devolve oitava da nota.
`Acc getAccidental()`	Devolve acidente da nota.
`boolean isSilence()`	Retorna `true` se e só se a nota representar silêncio (`Pitch.S`).
`String toString()`	Devolve representação textual da nota em linha com o esperado para o formato de representação de notas num ficheiro de texto (ver Secção 5).
`Note changeTempo(double f)`	Devolve nova nota com os mesmos atributos, mas duração multiplicada por factor `f`.
`Note octaveUp()`	Devolve nova nota com os mesmos atributos, mas para uma oitava acima.
`Note octaveDown()`	Devolve nova nota com os mesmos atributos, mas para uma oitava abaixo.

4 A classe `Melody`

A classe `Melody` permite armazenar e manipular sequências de notas, além de ter associado um título e autor. Para armazenar as notas internamente deverá **apenas usar um vector** ("array") de objectos `Note`. Além disso, **não pode usar métodos da classe** `java.util.Arrays`. Os métodos a seguir descritos deverão ser implementados.

Método(s)	**Descrição**
`Melody(String title, String author, int n)`	Constructor. Inicialmente, a melodia deverá codificar uma sequência de `n` notas, todas correspondentes a um silêncio de duração `0`.
`String getTitle()`	Devolve título.
`void setTitle(String title)`	Modifica título.
`String getAuthor()`	Devolve autor.
`void setAuthor(String author)`	Modifica autor.
`int notes()`	Devolve número de notas.
`double getDuration()`	Devolve duração da melodia, ou seja, a soma das durações das notas na melodia.
`Note get(int i)`	Devolve `i`-ésima nota, onde `i` poderá ir de `0` a `notes()-1`.
`void set(int i, Note n)`	Altera `i`-ésima nota para `n`, onde `i` poderá ir de `0` a `notes()-1`.
`void changeTempo(double f)`	Altera melodia, para que cada nota tenha duração multiplicada por factor `f`.
`void octaveUp()`	Altera melodia, subindo uma oitava para cada nota.
`void octaveDown()`	Altera melodia, descendo uma oitava para cada nota.
`void reverse()`	Inverte ordem das notas (primeira nota troca com última, segunda com penúltima, etc).
`void append(Melody other)`	Adiciona notas da melodia `other` ao fim desta melodia.

5 A classe `MelodyIO`

`MelodyIO` é uma classe utilitária com dois métodos, `load` e `save`, para respectivamente carregar e gravar uma melodia de/para um ficheiro. No código já é indicado (caso ache conveniente) como inicializar um objecto `Scanner` para leitura em `load`, e um objecto `PrintStream` para escrita em `save`.

Método(s)	**Descrição**
`static Melody load(File f)`	Carrega melodia do ficheiro `f`.
`static void save(File f, Melody m)`	Grava melodia `m` no ficheiro `f`.

Nota: Na leitura deverá ser conveniente o uso dos seguintes métodos estáticos definidos em `Pitch` e `Acc` respectivamente: `static Pitch valueOf(String s)` e `static Acc valueOf(String s)`.

O formato de texto das canções é ilustrado com o exemplo abaixo. O ficheiro começa com o título, autor e número de notas da melodia. A seguir aparecem representadas as notas na melodia; cada nota aparece no formato correspondente ao esperado para `Note.toString()`.

```
Happy Birthday
Patty Hill and Mildred J. Hill
27
0.25 S
0.25 D 4 NATURAL
0.25 D 4 NATURAL
0.5 E 4 NATURAL
0.5 D 4 NATURAL
0.5 G 4 NATURAL
1 F 4 SHARP
0.25 D 4 NATURAL
0.25 D 4 NATURAL
0.5 E 4 NATURAL
0.5 D 4 NATURAL
0.5 A 4 NATURAL
1 G 4 NATURAL
0.25 D 4 NATURAL
0.25 D 4 NATURAL
0.5 D 5 NATURAL
0.5 B 4 NATURAL
0.5 G 4 NATURAL
0.5 F 4 SHARP
1 E 4 NATURAL
0.25 C 5 NATURAL
0.25 C 5 NATURAL
0.5 B 4 NATURAL
0.5 G 4 NATURAL
0.5 A 4 NATURAL
1.5 G 4 NATURAL
0.25 S
```

6 Como iniciar o trabalho?

No pacote `pco.melody.test` existem alguns programas de teste que poderão ser úteis para orientar o trabalho na sua fase inicial. Sugerem-se os seguintes passos:

1. Execute o programa `DoReMiTest0` apenas para testar a reprodução de som;
2. Execute o programa `DoReMiTest1` para validar os aspectos básicos de funcionamento da classe `Note` – deverá ouvir uma melodia semelhante à do passo 1;
3. Execute o programa `DoReMiTest2` para validar os aspectos básicos de funcionamento da classe `Melody` – deverá ouvir uma melodia semelhante à dos passos 1 e 2.