Exame Recurso de "Introdução à Programação" (2009/2010)
9 de Fevereiro de 2010
Duração: 2:00H

Instruções muito importantes:

- Responda a cada grupo em folhas **separadas**.
- Verifique que **todas** as folhas estão identificadas com o número de caderno.
- 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 **`CourseRating`** cujos objectos representam a avaliação feita pelos alunos de um docente para uma dada disciplina. Cada objecto desta classe é caracterizado pela seguinte informação: a designação da disciplina, o nome do professor, o número de respostas com avaliação insuficiente, o número de respostas com avaliação suficiente, e o número de respostas com avaliação bom.

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

- **`public`** `CourseRating (`**`String`** `course,` **`String`** `professor)`**`:`** Constrói um objecto **`CourseRating`** para a disciplina `course` e o professor `professor`.

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

- **`public`** `String getProfessor()`**`:`** Devolve o nome do professor em avaliação.
- **`public`** `String getCourse()`**`:`** Devolve o nome disciplina em avaliação.
- **`public`** `int getNrAnswers()`**`:`** Devolve o número de respostas submetidas.
- **`public`** `void submitRating(char rating)`**`:`** Submete uma avaliação: 'I' para insuficiente, 'S' para suficiente e 'B' para bom.
- **`public`** `float overallRating()`**`:`** Devolve a classificação obtida (valor entre 1 e 3), a qual é calculada como a média ponderada das respostas submetidas, em que: insuficiente corresponde a 1 ponto; suficiente corresponde a 2 pontos; e bom corresponde a 3 pontos.
- **`public`** `String toString()`**`:`** Devolve uma `String` com os resultados da avaliação, nomeadamente na primeira linha o nome do professor e a designação da disciplina separados por uma vírgula, na linha seguinte as percentagens de respostas com classificação insuficiente, suficiente e bom, por esta ordem, separados por vírgulas. Por exemplo:

```
"Francisco Silva; POO
28.571428, 14.285714, 57.142857"
```

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

II. Para gerir as avaliações lectivas de um semestre é preciso definir uma classe **`SemesterRatings`** que representa um conjunto das avaliações dos docentes para as várias disciplina leccionadas. *Note que uma disciplina pode ser leccionada por vários docentes e cada docente pode leccionar várias disciplinas num semestre*. Cada elemento deste conjunto é representado por um objecto da classe **`CourseRating`**. O construtor da classe **`SemesterRatings`** é:

- **`public SemesterRatings`** `(`**`String`** `semester)`: Constrói um novo **`SemesterRatings`** que deve poder conter até `60` registos.

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

- **`public String`** `listGoodProfessors(float value)`**:** Devolve uma `String` com a lista com os nomes dos professores com avaliação média superior a `value` a uma dada disciplina, separados por ponto e vírgula; note que o mesmo professor pode aparecer várias vezes nesta lista.
- **`public boolean`** `addCourseRating(`**`CourseRating`** `c)` : Adiciona a avaliação `c` às avaliações do semestre, se possível, devolvendo **`true`**. Caso essa avaliação já exista, ou caso não se possam admitir mais avaliações, esta operação não faz nada e devolve **`false`**.
- **`public float`** `getCourseRating(`**`String`** `course)`**:** Devolve a classificação de uma disciplina; considere que a classificação da disciplina `course` é a média das classificações obtidas pelos vários docentes da disciplina, e que esta operação devolve `0.0f`, se não existir nenhuma classificação para a disciplina.
- **`public String`** `worstRatedCourse()`: Devolve o nome da disciplina com a pior classificação; em caso de empate, devolve o nome de uma disciplina qualquer, entre as disciplinas empatadas.

Programe a classe **`SemesterRatings`** em Java. Na resolução desta alínea considere que o método `searchIndexOf` já foi implementado:

- **`private int`** `searchIndexOf(`**`CourseRating`** `c)`**:** Devolve a posição no vector da avaliação `c` (devolve `-1` caso não exista);

III. Considere que pretende desenvolver uma aplicação de gestão de fotografias **AiFoto**. Neste exercício, não terá que programar, mas sim 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.

O **AiFoto** deverá:

- Permitir manter, numa biblioteca global, informação associada a cada foto armazenada.
- Permitir inserir e remover fotos da biblioteca global. Esta é a única maneira de alterar as músicas que **AiFoto** contém.
- Permitir criar e apagar eventos. Um evento (por exemplo, "aniversário da Matilde") é constituído por um subconjunto das fotos na biblioteca. Cada evento deve ter um nome. Deve ser possível criar até 256 eventos diferentes. Quando se remove um evento, as fotos que este continha devem continuar na estar na biblioteca global.
- Permitir adicionar e remover fotos de um evento (mantendo a foto na biblioteca global).

Nota: Só deve ser possível remover uma foto da biblioteca global se ela não pertencer a nenhum evento. Descreva (**sem implementar!**) a(s) classe(s) de que necessitaria para implementar o **AiFoto**. Ao fazê-lo, indique:

a) 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) Quais são as variáveis e constantes a declarar na classe. Para cada uma indique o seu tipo (e valor, no caso das constantes), e escreva uma pequena explicação sobre o que ela representa.

IV. Considere a seguinte classe

```
public class ReversibleList {

  private int contents[];
  private int SIZE;

  public ReversibleList(int size) {
    SIZE = size;
    contents = new int[size];
  }

  // Nas operações reverse, count e median, assuma sempre que
  // ReversibleList tem SIZE elementos.

  public void reverse() {
    // inverter a ordem dos elementos na lista contents
    // E.G. 9,2,4,5,2 -> 2,5,4,2,9
  }

  public int count(int x) {
    // devolver quantas ocorrências do valor x existem na lista
    // E.G. 2 em 9,2,4,5,2 -> 2
    // E.G. 5 em 9,2,4,5,2 -> 1
    // E.G. 7 em 9,2,4,5,2 -> 0
  }

  public int median(){
    // devolver a mediana dos valores da lista.
    // recorde que a mediana é o valor que separa a metade
    // superior da metade inferior na amostra ordenada
    // se SIZE for par, a mediana será a média (arrendondada) dos
    // dois valores medianos, de SIZE for impar, a mediana será
    // exactamente o valor no meio da amostra ordenada
    // E.G. 9,4,2,5,2 -> 4
    // E.G. 9,4,3,8,6,1 -> 5
  }
}
```

Programe:

a) O método `reverse`.

b) O método `count`.

c) O método `median`.