2º Teste de "Introdução à Programação" 2008-09
9 de Janeiro de 2009

Duração: 2:00H

Instruções importantes:

- **Responda a todos os grupos em folhas separadas.**
- Antes de começar a resolver um exercício, leia calmamente 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.
- Mais: recorde o Código de Ética.
- Não é permitido sair da sala antes que o teste termine.

I. Pretende-se programar, em Java, uma classe **`Friend`** cujos objectos representam amigos num site de comunidade virtual (como o hi5). Cada amigo deve registar a seguinte informação: um nome, o sexo, a idade, o país, e um ranking (1-5). 5 significa muito amigo, 0 significa vagamente conhecido. O sexo é indicado por 1 (masculino) e 2 (feminino).

Os construtores da classe **`Friend`** estão definidos da seguinte forma:

- **`public`** `Friend(String name, int sex):` Constrói um amigo com o nome `name`, ranking 0, país "Portugal" e sexo `sex` .
- **`public`** `Friend(String name, int rank, String country, int sex):` Constrói um jogador com o nome `name`, ranking `rank`, país `country` e sexo `sex`.

A interface pública da classe **`Friend`** está definida da seguinte forma:

- **`public`** `String getName():` Devolve o nome do amigo;
- **`public int`** `getCountry():` Devolve o país do amigo;
- **`public int`** `getRank():` Devolve o ranking do amigo;
- **`public void`** `raiseRank():` Aumenta o ranking do amigo em uma unidade, excepto se já tiver o ranking máximo;
- **`public void`** `lowerRank():` Diminui o ranking do amigo em uma unidade, excepto se já tiver o ranking mínimo.
- **`public boolean`** `isGirl():` Verifica se o amigo é uma rapariga.
- **`public boolean`** `equals(Friend f):` indica se dois objectos **`Friend`** são iguais, ou seja, se representam a mesma pessoa, o que acontece se são caracterizados exactamente pela mesma informação.

Programe, em Java, a classe **`Friend`**. Não se esqueça das constantes e variáveis de instância (memória ou estado) da classe.

II. Para construir uma aplicação de comunidade virtual é preciso definir uma classe **`Friendship`** que representa um conjunto de vários amigos (vulgo, a "malta"). Cada amigo é representado por um objecto da classe **`Friend`**. (**dica**: na classe **`Friendship`** use um vector para guardar os vários amigos).

Importante: quando se fala aqui de amigos, usamos as palavras "amigo" e "amigos" para designar indiferentemente amigos de qualquer sexo. Quando nos queremos referir a amigos de um sexo particular, usamos a qualificação necessária, por exemplo "amigo masculino" ou "amiga".

O construtor da classe `Friendship` é o seguinte:

- **`public`** `Friendship(`**`int`** `n)`: Constrói um objecto da classe **`Friendship`**. O parâmetro `n` indica o número máximo de amigos que pode admitir.

A interface pública da classe **`Friendship`** é a seguinte:

- **`public void`** `addFriend(String name, int rank, String country, int sex)` : Adiciona um novo amigo à malta, caso seja possível. Caso o amigo já exista na malta ou não seja possível acomodar mais amigos, esta operação não faz nada.
- **`public String`** `countFriends()`: Devolve o número total de amigos na malta;
- **`public String`** `getBestFriend()`: Devolve o nome do amigo com maior ranking. Em caso de empate, devolve o nome de um dos melhores amigos. Se não existir nenhum amigo devolve uma String vazia;
- **`public double`** `getAverageBoyFriends()`**:** Devolve a média dos rankings dos amigos masculinos, ou -1 se não existirem tais amigos;
- **`public int`** `countFriendsIn(String country)`: Devolve o número total de amigos que são do país `country`;
- **`public boolean`** `uniqueFriend(String name)`. Indica se existe um único amigo com o nome `name`. Devolve `false` quer o amigo não exista, quer ocorra mais do que uma vez.

a) Programe, em Java, a classe **`Friendship`**. Não se esqueça das variáveis de instância (memória ou estado) da classe.

b) Escreva um programa principal em Java que leia, do ficheiro de texto "friends.txt", a informação sobre um conjunto de amigos e escreva a seguinte informação na consola:

- Número de amigos;
- Nome do melhor amigo;
- Média dos rankings dos amigos masculinos;
- Escreva, para cada sexo, o nome dos amigos desse sexo.

O ficheiro de texto tem o seguinte formato: na primeira linha aparece um número inteiro positivo que indica o total de amigos. Nas linhas seguintes aparecem as informações dos amigos. Para cada amigo, aparece, em linhas separadas, o nome do amigo, o sexo, o país, e o ranking. Por exemplo, um ficheiro com 3 amigos terá 13 linhas.

III. Dada uma sequência finita e estritamente crescente de números naturais (>=0)

n_1, n_2, n_3, n_4, n_5, n_6, ...

as diferenças (n_{i+1} - n_i) entre elementos consecutivos da sequência são sempre números estritamente positivos.

Numa tal sequência (finita) a **maior** diferença entre elementos consecutivos chama-se "o maior degrau" da sequência. Por exemplo, na sequência

0, 1, 2, 4, 5, 8, 10, 11, 14

o maior degrau é 3 (14-11 e 8-5).

Considere o seguinte fragmento da classe Java **`Sequence`**, definida da forma seguinte

```
class Sequence {
  int elements[];
  int size;

  Sequence(int thesize) {
     size = 0;
     elements = new int[thesize];
  }

  void addElement(int elt) {
     elements[size] = elt;
     size = size + 1;
  }

}
```

a) Considerando apenas a implementação acima, indique, justificando, que situação pode causar o abortar do programa quando o método `addElement` é chamado várias vezes num objecto previamente construído com `Sequence(int thesize)`.

b) Implemente na classe `Sequence` acima um novo método `maxStep()` que devolve o maior degrau da sequência.

c) Implemente na classe `Sequence` acima um novo método `trim()` que divide (divisão inteira) por 2 (dois) todos os elementos da sequência, mas **só** se a sequência resultante continuar a ser estritamente crescente. Se tal não for possível, o método `trim()` não faz nada.