Programação Orientada pelos Objectos

Exame de Recurso
2009/07/07

Atenção: A fraude numa prova de avaliação é punida com a reprovação.

Primeiro grupo

Na informática trabalhamos frequentemente, por exemplo quando usamos bases de dados, com entidades que têm um índice único, muitas vezes designado de "chave". Considere a seguinte interface que representa a obrigatoriedade de ter um índice inteiro:

```
public interface Indexable
{
   public int index();
}
```

Nesta altura do ano andamos todos (alunos e professores) preocupados com as notas. Considere a seguinte classe *Grade* que representa a nota obtida por um aluno numa dada disciplina com duas componentes de avaliação: laboratórios (40% do resultado final) e exame (60% do resultado final).

```
public class Grade implements Comparable<Grade>, Indexable
{
   private int studentNumber;
   private double labs;
   private double exam;
  …
}
```

Programe as seguintes operações da classe *Grade*:

a) O construtor *public Grade(String s)* que recebe uma String contendo três campos: nº de aluno, nota dos laboratórios e nota do exame. As notas são apresentadas com uma casa decimal. Cada campo é separado do seguinte por um espaço.

b) A operação *int result()* que devolve o resultado final na disciplina. Pressupõe-se que só estamos a processar alunos que foram aprovados na componente laboratorial. Se a nota do exame for inferior a 7.5 o resultado é a nota do exame; caso contrário é uma soma ponderada das componentes de laboratório (40% do resultado final) e exame (60% do resultado final).

c) A operação *String toString()* que deverá construir e devolver uma linha da pauta com o nº de aluno, nota do laboratório, nota do exame, resultado final e ainda um "R" se o aluno estiver reprovado. Os vários campos devem ser formatados para que as barras verticais separadoras apareçam alinhadas quando se mandar escrever toda a pauta. A seguir são mostradas duas linhas de exemplo:

```
15605 | 11,0 |  6,9 |  7 |  R
15832 | 13,2 | 16,8 | 15 |
```

d) As operações relativas à implementação das duas interfaces. A comparação é efectuada exclusivamente com base no resultado final. O índice corresponde ao nº de aluno. Recorde que a interface *Comparable<T>* declara a operação *public int compareTo(T)*.

Segundo grupo

Considere a classe *Marks*, que representa uma pauta, cujo esqueleto se fornece de seguida:

```
public class Marks
{
    public class Indexed<N extends Indexable>
                                implements Comparable<N>
    {
      private N value;

      public Indexed(N initialValue){value = initialValue;}
      public int getIndex()          {return value.index();}
      public N getValue()            {return value;}
      public String toString()       {return value.toString();}
      public int compareTo(N other)
                  {return value.index() - other.index();}
     }

   private ArrayList<Indexed<Grade>> grades;

   public Marks() {…}
   public ArrayList<Indexed<Grade>> getGrades(){…}
   public void read(Scanner in) {…}
   public double averageGrade(){…}
   public Grade maxGrade(){…}
   public void sort() {…}
}
```

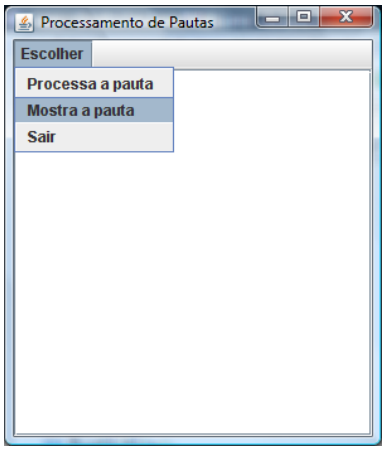
Implemente as seguintes operações da classe *Marks*:

a) O construtor que inicializa o *ArrayList* de notas indexadas;
b) O selector *getGrades* que devolve o *ArrayList*;

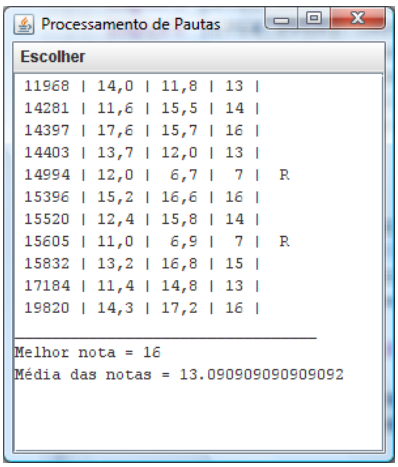
c) O modificador *read* que lê do scanner fornecido e preenche o *ArrayList*;
d) O selector *averageGrade* que devolve o valor da média das notas da pauta;
e) O selector *maxGrade* que devolve a linha da pauta (instância da classe *Grade*) correspondente à melhor nota constante nessa pauta;
f) O modificador *sort* que ordena o *ArrayList*. Nesta alínea poderá escolher livremente o método de ordenação, desde que seja crescente e de acordo com o índice já definido.

Terceiro grupo

Implemente a classe *Main* para efectuar o processamento das notas, que deve gerar um menu simples como o representado na figura. Quando se carrega na opção "Processa a pauta" é lido o ficheiro *data/Grades.txt* que contem os dados em bruto da pauta. Cada linha do ficheiro tem o formato descrito na alínea a) do grupo 1.



Quando se carrega na opção "Mostra a pauta", é mostrada a mesma, seguida de informação sobre a melhor nota e a média das notas, tal como representado na figura seguinte.



Quando se carrega na opção "Sair", o programa termina, como seria esperado ☺.

Deve usar a seguinte estrutura de dados:

```
public class Main
{
   private static Marks allGrades;
   private static JFrame iface;
   private static JTextArea texto;
   private static JMenuBar menu;
…
```

Quarto grupo

Um **anagrama** (do grego *ana* = "voltar" ou "repetir" + *graphein* = "escrever") é um rearranjo das letras de uma palavra para produzir outras palavras, utilizando todas as letras originais exactamente uma vez. Pretende-se que escreva um programa **recursivo** que escreva todos os anagramas de uma palavra fornecida. O algoritmo é relativamente simples, como verá.

Dada uma palavra de N letras, podemos pensar numa operação que sabe calcular todas as palavras de N-1 letras prefixadas (iniciadas) pela letra que ficou de fora. Fazemos isto num ciclo em que deixamos de fora cada um dos caracteres da palavra inicial. Se prefixarmos as permutações das N-1 letras com a letra que ficou de fora, então obtemos todos os anagramas da palavra original. O algoritmo repete-se recursivamente até atingir uma palavra com apenas uma letra. Em cada chamada recursiva vai-se passando o prefixo usado na chamada anterior (que vai crescendo uma letra de cada vez) e a palavra constituída pelas letras remanescentes (aquelas que ainda é preciso permutar). Quando só existe uma letra remanescente, não há mais permutações a fazer e por isso basta escrever a palavra constituída pelo prefixo mais essa letra e a recursão termina.

Exemplo da utilização da classe *Anagrams*:

```
Indique a palavra a anagramar: boa
boa, bao, oba, oab, abo, aob

Indique a palavra a anagramar: ana
ana, aan, naa, naa, aan, ana
```

a) Crie a classe *Anagrams* e a sua operação *main* que pede uma palavra ao utilizador e faz a chamada inicial à operação *printAnagrams* cuja interface é descrita na alínea seguinte.

b) Escreva o corpo da operação recursiva *void printAnagrams(String prefix, String remain)*. Use a operação *substring* cuja descrição retirada do *Javadoc* da classe *String* se fornece a seguir.

public String ***substring****(int beginIndex, int endIndex)*
Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` *and extends to the character at index* `endIndex-1`.
Example: "extravaganza".substring(2, 7)) => "trava"