Universidade Federal de Uberlândia
Faculdade de Computação
FACOM39502 – Programação Orientada a Objetos
2023/2
Prof. Renato Pimentel

Laboratório 10 – Arquivos

Atividade individual.

1 Introdução

Nesta prática, resolveremos um exercício sobre arquivos e exceções. Em aulas anteriores, foram vistos os conceitos de *array* e de herança. Um *array* (vetor ou matriz) é um *objeto* que aponta para um conjunto de dados de tipos primitivos, ou para um conjunto de outros objetos (vetor de objetos). A herança permite relacionar as classes de um programa hierarquicamente, definindo relações do tipo "é um"/"é um tipo de".

Os *arquivos* permitem que o programa acesse e envie dados externos. O Java permite lidar com todos os tipos de entrada e saída através de uma abstração conhecida como *stream* (canal, fluxo de dados). *Streams* são unidirecionais e podem ser de dois tipos:

- *Leitura/Entrada*: para obter informações externas;
- *Escrita?Saída*: para enviar informações.

O pacote `java.io` define um grande número de classes para ler e escrever *streams*. As classes são divididas em dois grupos, baseadas no tipo de dados sobre os quais operam:

- `InputStream` e `OutputStream`: E/S de *bytes* (arquivos binários);
- `Reader` e `Writer`: E/S de caracteres (`char`) e de caracteres Unicode (arquivos de texto).

Um outro conceito importante quando se lida com arquivos é o de "Persistência de dados", que consiste no armazenamento *confiável* e *coerente* das informações em um sistema de armazenamento de dados. Ela pode ser implementada por meio de duas abordagens:

- Armazenar os dados em arquivos de texto;
- Usar *serialização*, permitindo gravar objetos em arquivos binários.

Uma *exceção* em Java é um sinal gerado em tempo de execução do programa, que é comunicado ao mesmo indicando a ocorrência de um erro recuperável. No uso de arquivos, por exemplo, devemos implementar o código num bloco *try/catch*, pois exceções podem ser geradas (`IOException`). Uma vez que se tratam, neste caso, de *exceções verificadas*, seu tratamento é requisito para a compilação do código-fonte.

A seguir são apresentados exemplos da implementação do uso de arquivos, vista na aula teórica. São duas classes:

- `UsarArquivo`, que contém o programa principal;
- `GerenciamentoArquivos`, que contém os métodos para a leitura e escrita de dados em um arquivo.

```
1  public class UsarArquivo {
2     public static void main( String[] args ) {
3        String nome[]=new String [3];
4        int idade[]=new int [3];
5        double nota[] = new double [3];
6        nome[0] = "José";  nome[1] = "Márcia"; nome[2] = "Carla";
7        idade[0] = 23;    idade[1] = 20;      idade[2] = 18;
8        nota[0] = 7.5;      nota[1] = 7;       nota[2] = 8.5;
9        GerenciamentoArquivos gerente = new GerenciamentoArquivos();
10       gerente.escrita( "teste.txt", nome, idade, nota );
11       gerente.leitura( "teste.txt" );
12    }
13 }
```

```
1  import java.io.*;
2  public class GerenciamentoArquivos {
3
4     public void escrita(String nomeArq, String[] vet1, int[] vet2,
      double[] vet3) {
5
6        try {
7           FileWriter arq = new FileWriter(nomeArq);
8           PrintWriter out = new PrintWriter(arq);
9
10          for (int i = 0; i < vet1.length; i++) {
11             String linha = vet1[i] + ":" + vet2[i] + ":" + vet3[i
      ];
```

```
12              out.println(linha);
13           }
14           out.close();
15        } catch (IOException erro) {
16           System.out.println("Erro na escrita dos dados");
17        }
18     }  //fim do método escrita()
19  }
```

2 Exercício

1. Em um campeonato de futebol, quatro times (A, B, C e D) jogam partidas no esquema turno-returno. Para dois times A e B, sendo o turno disputado no campo de A, o returno é disputado no campo de B. Os resultados dos jogos encontram-se armazenados em um arquivo, como segue:

   `*;v;e;v`

   `e;*;e;d`

   `v;v;*;d`

   `e;v;e;*`

   Este arquivo pode ser entendido como uma matriz, da seguinte forma:

$$\begin{pmatrix} & A & B & C & D \\ A & * & v & e & v \\ B & e & * & e & d \\ C & v & v & * & d \\ D & e & v & e & * \end{pmatrix}$$

   O símbolo `v` indica vitória do time da casa, o símbolo `e` indica empate e o símbolo `d` indica derrota do time da casa. Assim, cada linha apresenta os resultados de um time em seu campo. Ou seja, a primeira linha corresponde aos resultados do time A quando ele jogou em seu campo. Temos assim que: A venceu B, empatou com C e venceu D, jogando em casa. As colunas representam então os resultados de cada time nos campos dos adversários, porém nesse caso o símbolo `v` significa derrota no campo do adversário (igual a vitória do time da casa) e o símbolo `d` significa vitória no campo do adversário (igual derrota do time da

casa). Para a primeira coluna, temos: A empatou com B, perdeu de C e empatou com D.

Faça um programa em Java que leia essa matriz do arquivo, armazenando-a em uma matriz, e forneça o número total de pontos obtido por cada time no campeonato (turno e returno) em um arquivo chamado `resultados.txt`. Não é necessário fornecer a classificação final dos quatro times. Considere que cada vitória vale 3 pontos no campeonato, os empates valem um ponto cada e as derrotas não somam pontos. O programa deve pedir para o usuário os nomes dos times A, B, C e D.

Dicas:

Usando o conceito de persistência, os dados encontram-se separados pelo separador `;`. Na leitura de cada linha do arquivo, o comando split pode então ser usado na separação dos campos presentes. Segue abaixo um exemplo envolvendo o uso do comando split (com outro tipo de delimitador).

```
class BuscaPalavrasEmUmaString {
   public static void main (String args[])  {
      String texto = "Isto#é#um#texto#de#teste";
      // usando delimitador #
      String result[]  = texto.split("#");
      for (int i=0; i<result.length; i++)
         System.out.println (i+1+": "+result[i]);
   }
}
```

O aplicativo deve ser composto por duas ou três classes. Uma para o programa principal, uma para a manipulação dos arquivos e outra para a manipulação dos cálculos de pontos do campeonato, sendo que esta pode ser dispensada se desejado.

Materiais complementares

- Conteúdo de arquivos: https://www.facom.ufu.br/~rpimentel/files/facom39502-2023-2/facom39502.arquivos.pdf
- https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html (em inglês)
- https://www.caelum.com.br/apostila/apostila-java-orientacao-objetos.pdf (em português)