FICHA 09 – FICHEIROS

Os ficheiros são o mecanismo que permite guardar dados de modo persistente em disco.

1. A classe File

Qualquer ficheiro para ser usado tem que ser identificado de modo inequívoco. Os objectos da classe **File** permitem representar o nome de um ficheiro em Java, de forma independente do Sistema Operativo usado.

A classe File pertence à package **java.io**, e pode-se criar um objecto do tipo File, fornecendo ao construtor uma string com o nome do ficheiro:

```
import java.io.*;
(…)
 File f1;
 f1 = new File("listaDeCompras");
```

O ficheiro não tem que existir à partida, nem passa a existir apenas por se ter criado o objecto do tipo File. A única coisa criada é uma representação (para o Java) de um nome, para um hipotético ficheiro (ou directoria).

No entanto, se o ficheiro existir o objecto File criado realmente refere-se ao mesmo. Por exemplo, podemos apagar um ficheiro ou mudar-lhe o nome da seguinte forma:

```
File um = new File("temporario");
File outro = new File("lixo");
um.renameTo(outro); //O temporario passa a chamar-se lixo
outro.delete();     //O lixo é apagado!
```

Alguns métodos da classe File:

public boolean **renameTo(File dest)** throws SecurityException	Muda o nome do ficheiro (ou directoria) em questão para "dest".
public boolean **delete()** throws SecurityException	Apaga o ficheiro (ou directoria) em questão. Caso seja uma directoria, deve estar vazia.

2. Escrita em Ficheiros de texto

Escrever num ficheiro implica de algum modo criar uma ligação entre a nossa aplicação e esse ficheiro. Essa ligação permitirá transferir a informação entre uma e outro. Essas ligações ou "condutas" são por vezes denominadas de streams.

A classe FileWriter

A classe **FileWriter**, pertencente à package java.io, permite criar uma ligação entre um ficheiro e uma aplicação, ou seja, permite que seja trocada informação entre a aplicação e um ficheiro.

Pode-se criar uma instância da FileWriter à custa de um objecto File. A tentativa de criação de um objecto do tipo FileWriter pode dar origem a uma excepção do tipo **IOException**, caso o ficheiro seja impossível de abrir ou de criar. Esta excepção deve ser apanhada, ou declarada pelo método onde o objecto está a ser criado.

```
import java.io.*;
(…)
  try {
    File f = new File("resultad.out");
    FileWriter fs = new FileWriter(f);
  (…)
  } catch (IOException e) {
    System.out.println(
      "Ocorreu uma excepção " + e + " ao criar o FileWriter fs");
  }
```

No exemplo acima, passa a haver uma ligação entre a aplicação e um ficheiro de nome "resultad.out". Mesmo que o ficheiro não exista inicialmente, é criado um ficheiro novo com esse nome e a ligação estabelece-se.

No entanto a ligação obtida à custa de FileWriter trabalha a informação em bytes o que se torna algo inconveniente (por exemplo não reconhece formatações de texto: mudanças de linha, tabs, etc). Ou seja, esta classe fornece-nos uma ligação com um ficheiro, mas não nos fornece métodos práticos para enviarmos a informação que desejamos.

A classe PrintWriter

As instâncias da classe **PrintWriter**, pertencente também à pakage java.io, dispõem de métodos que reconhecem linhas e strings. Para escrever comodamente num ficheiro, será necessário converter a instância da classe FileWriter, (que efectua a ligação ao ficheiro), numa instância da classe PrintWriter (que dispõe dos métodos de que necessitamos). A conversão pode ser efectuada do seguinte modo:

```
import java.io.*;
(…)
    try {
      File f = new File("fich.txt");
      FileWriter fs = new FileWriter(f);
      PrintWriter ps = new PrintWriter(fs);
      ps.println("linha1 !!!");
      ps.println("linha2 !!!");
      ps.close();
    } catch (IOException e) {
      System.out.println(
        "Ocorreu uma excepção " + e + " ao criar o FileWriter fs");
    }
```

No exemplo acima cria-se um novo ficheiro (ou abre-se, se já existir), escrevem-se 2 linhas e fecha-se de novo. Caso o ficheiro já existisse o seu conteúdo seria substituído pelas duas linhas em questão.

Normalmente, as instâncias das classes File e FileWriter são apenas usadas para criar a PrintWriter, pelo que as variáveis não são necessárias, podendo abreviar-se para:

```
PrintWriter ps = new PrintWriter(new FileWriter(new File("fich.txt")));
```

3. Leitura de Ficheiros de texto

De um modo análogo à escrita de dados em ficheiros, também é possível obter-se dados de ficheiro. A leitura de um ficheiro de texto parte de uma FileReader (obtida de uma File) que modela um stream de entrada. Caso se tente criar uma FileReader de um ficheiro inexistente, é lançada uma **FileNotFoudException**. De seguida, essa FileReader é usada para criar um InputStreamReader que modela uma ligação entre a fonte de dados e a aplicação. Embora trabalhe com caracteres (em vez de bytes), não conhece mudanças de linha. Assim deve-se usar para criar um **BufferedReader**. Esta classe fornece o método **readLine** que nos permite ler a entrada linha a linha.

O código seguinte abre o ficheiro "fich.txt" para leitura, lê as duas primeiras linhas e mostra-as no monitor.

```
try  {
     File f = new File("fich.txt");
     FileReader fis = new FileReader(f);
     BufferedReader br = new BufferedReader(isr);
     String s = br.readLine();
     System.out.println("1: " + s);
     s = br.readLine();
     System.out.println("2: " + s);
   } catch (FileNotFoundException e) {
     System.out.println("Ficheiro inexistente fich.txt");
   } catch (IOException e) {
     System.out.println("Excepção ao ler uma linha do ficheiro");
   }
```

4. Ficheiros de objectos

De forma análoga à escrita/leitura de texto também se pode fazer escrita/leitura de objectos em ficheiro, mas, nestes casos, devem ser utilizadas as classes

FileInputStream e **FileOutputStream** para estabelecer os fluxos de dados correspondente:

```
FileInputStream is = new FileInputStream(new File(nomeDoFicheiro));
FileOutputStream os = new FileOutputStream(new File(nomeDoFicheiro));
```

Ou, abreviadamente:

```
FileInputStream is = new FileInputStream(nomeDoFicheiro);
FileOutputStream os = new FileOutputStream(nomeDoFicheiro);
```

Para se armazenar em ficheiro objectos de classes definidas pelo programador, terá que se acrescentar no cabeçalho dessas classes as palavras **implements Serializable**:

```
public class Turma implements Serializable
```

Agora é possível criar objectos para manipular os ficheiros de objectos:

```
ObjectInputStream ois = new ObjectInputStream(is);
ObjectOutputStream oos = new ObjectOutputStream(os);
```

5. Excepções

Uma excepção é algo que ocorre durante a execução de um método e que o impede de funcionar de acordo com o previsto. A sua detecção e tratamento permite lidar com essas situações de forma controlada.

Os métodos que utilizam operações de entrada e saída, têm que tratar as excepções (instrução **try – catch**) ou declarar a sua propagação à custa da cláusula **throws**.

Por exemplo no método abreLeitura(), que abre um ficheiro para leitura, detecta-se a inexistência do ficheiro:

```
public boolean abreLeitura(String nomeDoFicheiro)
```

```
{
  try {
    iS = new ObjectInputStream(new FileInputStream(nomeDoFicheiro));
    return true;
  } catch (IOException e) {
    return false;
  }
}
```

O método tenta executar as instruções incluídas no bloco **try**. Se não for gerado qualquer erro, a sua execução prossegue na instrução seguinte ao bloco **catch**. No caso de ocorrer um erro (excepção), os blocos catch são verificados, procurando-se o primeiro que se aplique ao tipo de excepção que surgiu. No método abreLeitura(), a excepção criada (FileNotFoundException, pois o ficheiro não existe) é do tipo IOException, pelo que se enquadra no catch. Assim sendo, a instrução do catch é executada, terminando em seguida o método e devolvendo false, para indicar que não foi possível abrir o ficheiro.

6. A classe FicheiroDeObjectos

Pode ser desenvolvida uma classe que simplifique a utilização de ficheiros de objectos que inclua métodos para abrir um ficheiro, escrever um objecto num ficheiro, ler um objecto a partir de um ficheiro e fechar um ficheiro. De modo análogo pode ser desenvolvida uma classe para utilizar ficheiros de texto.

A classe terá como atributos referências para um objecto da classe **ObjectInputStream** e para um objecto da classe **ObjectOutputStream**:

```
import java.io.*;
public class FicheiroDeObjectos
{
  private ObjectInputStream iS;
  private ObjectOutputStream oS;
}
```

Os métodos de abertura do ficheiro para leitura e para escrita podem ser:

```
public void abreLeitura(String nomeDoFicheiro)
{
  try {
    iS = new ObjectInputStream(new FileInputStream(nomeDoFicheiro));
  } catch (Exception e) {
    System.out.println ("Não foi possível abrir o ficheiro para leitura");
  }
}
```

```
public void abreEscrita(String nomeDoFicheiro)
{
  try {
    oS = new ObjectOutputStream(new FileOutputStream(nomeDoFicheiro));
  } catch (Exception e) {
    System.out.println ("Não foi possível abrir o ficheiro para escrita");
  }
}
```

A leitura de um objecto a partir de um ficheiro pode ser efectuada através do método **readObject**() da classe ObjectInputStream:

```
public Object leObjecto()
{
  try {
    return iS.readObject();
  } catch (ClassNotFoundException e) {
    System.out.println ("O ficheiro não contém objectos");
  }
  catch (Exception e1) {
    System.out.println ("Não foi possível ler o ficheiro");
  }
}
```

Uma vez que se pretende que o método possa ler qualquer tipo de objecto, não se pode utilizar o nome de nenhuma classe em particular, mas caberá ao método que o chamar concretizar, através de um cast, qual a classe a que esse objecto deve pertencer.

O método pode gerar uma excepção do tipo **ClassNotFoundException** que será gerada se o ficheiro não contiver objectos.

A escrita de um objecto num ficheiro pode ser obtida com o método:

```
public void escreveObjecto(Object o)
{
  try {
    oS.writeObject(o);
  } catch (Exception e) {
    System.out.println ("Não foi possível abrir escrever no ficheiro");
  }
}
```

Os métodos para fechar os ficheiros limitam-se a utilizar o método **close**() das classes ObjectInputStream e ObjectOutputStream:

```
public void fechaLeitura()
{
  try {
    iS.close();
  } catch (Exception e) {
    System.out.println ("Não foi possível abrir fechar o ficheiro");
  }
}
```

```
public void fechaEscrita()
{
  try {
    oS.close();
  } catch (Exception e) {
    System.out.println ("Não foi possível abrir fechar o ficheiro");
  }
}
```

Exercícios

1. Cópia

Escreva um programa que faça uma cópia de um ficheiro de texto, mudando-lhe a extensão para ".bkp". O nome do ficheiro a copiar deve ser introduzido pelo utilizador.

2. Temperaturas

Crie um programa que peça ao utilizador o nome de um ficheiro de texto contendo as temperaturas mínimas e máximas de uma determinada cidade ao longo de um mês. Escreva um programa que leia esse ficheiro e determine a maior amplitude térmica verificada (diferença entre a máxima e a mínima). O programa deve ainda indicar em que dia do mês se verificou.

3. Permutações

Crie um programa que peça ao utilizador o nome de um ficheiro. Esse ficheiro deve conter duas linhas: a primeira linha deve conter uma palavra de quatro letras, e a segunda linha uma palavra até oito letras. Utilize a primeira linha para fazer permutações com as letras da palavra, e use a segunda como o nome do ficheiro onde irá guardar essas permutações. A palavra original e as permutações devem ainda ser mostradas no monitor.

4. Gestão de contactos

No problema da aula anteriro sobre um sistema de gestão de contactos faça as alterações necessárias para que:

- - ao entrar seja toda a informação carregada de ficheiro(s) de objectos e
- - ao sair seja toda a informação armazenada em ficheiro(s) de objectos.

5. Teste americano

As respostas a um teste do tipo americano, contendo 20 perguntas de escolha múltipla, estão armazenadas num ficheiro de objectos adequado. Cada resposta pode tomar o

valor 1, 2, 3, 4 ou 5, correspondente à opção seleccionada, e, no caso de não ter sido respondida, o valor 0. Crie um programa que leia um destes ficheiros e determine para uma turma (30 alunos) a classificação obtida por cada um – a chave com as respostas correctas é fornecida através de outro ficheiro constituído por números inteiros. Uma resposta certa vale 1 ponto, uma resposta errada desconta 0.5 pontos e uma resposta em branco não tem qualquer influência no resultado final.