

Ficheiros de texto

- Ficheiros de texto
- Escrita de informação em ficheiros de texto
- Leitura do conteúdo de ficheiros de texto
- Exemplos



Introdução

- Em todos os programas desenvolvidos até ao momento, a informação manipulada era perdida sempre que terminamos os programas.
- Isto deve-se ao facto de as variáveis que declaramos reservarem espaço na memória do computador, que depois é libertada quando o programa termina.
- Para armazenarmos permanentemente informação gerada pelos nossos programas, temos que a guardar no disco rígido do computador (ou em qualquer outro dispositivo de memória de massa).
- Isto é possível através da utilização de ficheiros.
- Nesta aula estamos apenas interessados em estudar a utilização de ficheiros de texto.

Ficheiros e Directórios

- O que é um ficheiro?
 - Estrutura de armazenamento de informação;
 - Uma sequência de ``0" e ``1" armazenados (informação binária).
- O que é um directório?
 - Tipo especial de ficheiro que armazena uma lista de referências a ficheiros.
- Características:
 - Localização no sistema de ficheiros (directório e nome);
 - Têm a si associadas permissões de leitura, escrita e execução.

Utilização de ficheiros em JAVA

- Classe `File` (`java.io.File`)
- Permite:
 - Confirmar a existência de ficheiros;
 - Verificar e modificar as permissões de ficheiros;
 - Verificar qual o tipo de ficheiro (directório, ficheiro normal, etc.);
 - Criar directórios;
 - Listar o conteúdo de directórios;
 - Apagar ficheiros.
 - ...



Ficheiros de texto

- Os dados são interpretados e transformados de acordo com formatos de representação de texto;
- Cada carácter é codificado (ASCII, Unicode, UTF-8, ...)
- Texto em Java:
 - Os tipos `char` e `String` codificam o texto com a codificação unicode 16 bits;
 - Esse detalhe de implementação do Java é, no entanto, transparente para o programador;
 - Os métodos (funções) de entrada/saída fazem automaticamente a tradução de ou para a codificação escolhida;
 - Existem também constantes literais para alguns caracteres especiais:
 - `'\n'`: nova linha;
 - `'\t'`: tabulação horizontal;
 - `'\"'`: carácter `"` , ...

Escrita de ficheiros em Java

- Classe `PrintWriter` (`java.io.PrintWriter`);
- Interface similar à do `PrintStream` (`System.out`);
- Utilização:
 - Criar uma entidade (objecto) `File` associada ao nome do ficheiro desejado:

```
File fout = new File(nomeFicheiro);
```

- Declaração e criação de um objecto tipo `PrintWriter` associado a esse objecto tipo `File`:

```
PrintWriter pwf = new PrintWriter(fout);
```

- Escrever sobre o ficheiro:

```
pwf.println(line);
```

- Fechar o ficheiro

```
pwf.close();
```



E quando a utilização falha?

- Operações sobre um `PrintWriter` podem falhar imprevisivelmente!
- Para lidar com esse tipo de situações a linguagem Java utiliza uma aproximação defensiva gerando (*checked*) excepções;
- A classe `PrintWriter` da biblioteca Java obriga o programador a lidar explicitamente com a excepção: `IOException`.
- Nos operações de abertura de ficheiros (não só na classe `PrintWriter`, mas também na classe a utilizar para leitura de ficheiros) é necessário lidar explicitamente com este tipo de excepções (**throws `IOException`**)

Leitura de ficheiros em Java

- Tipo de dados `Scanner` (`java.util.Scanner`);
- Em vez do `System.in` associar o `Scanner` ao ficheiro a ler;
- Utilização:

- Criar um objecto `File` associada ao nome do ficheiro desejado:

```
File fin = new File(nomeFicheiro);
```

- Declaração e criação de um objecto tipo `Scanner` associado a esse objecto tipo `File`:

```
Scanner scf = new Scanner(fin);
```

- Ler do ficheiro:

```
while(scf.hasNextLine())  
    String line = scf.nextLine();
```

- Fechar o ficheiro:

```
scf.close();
```



Exemplo

```
import java.io.*;
```

```
public class Ficheiros {  
  
    public static void main(String[] args) throws IOException{  
  
        Scanner kb = new Scanner(System.in);  
        System.out.print("Ficheiro de entrada: ");  
        String nameIn = kb.nextLine();  
  
        File fin = new File(nameIn);  
  
        Scanner scf = new Scanner(fin);  
        System.out.print("Ficheiro de saida: ");  
        String nameOut = kb.nextLine();  
  
        File fout = new File(nameOut);  
  
        PrintWriter pw = new PrintWriter(fout);  
  
        while(scf.hasNextLine())  
            pw.println(scf.nextLine());  
  
        scf.close();  
  
        pw.close();  
    }  
}
```



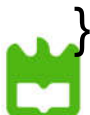
Exemplo, class FileWriter

```
public static void main(String[] args) throws IOException{
    Scanner kb = new Scanner(System.in);
    System.out.print("Ficheiro de entrada: ");
    String nameIn = kb.nextLine();
    File fin = new File(nameIn);
    Scanner scf = new Scanner(fin);
    System.out.print("Ficheiro de saida: ");
    String nameOut = kb.nextLine();
    // Em lugar da classe File pode usar-se a classe FileWriter
    // Tem a vantagem de abrir ficheiros em modo append (true)
    FileWriter fout = new FileWriter(nameOut, true);
    PrintWriter pw = new PrintWriter(fout);
    while(scf.hasNextLine())
        pw.println(scf.nextLine());
    scf.close();
    pw.close();
}
```



Class File, exemplos de funções

```
String nomeIn = "notas.txt"
File fin = new File(nomeIn);
if (!fin.exists()) {
    System.out.println("ERROR: input file " + nomeIn + " does not exist!");
} else if (fin.isDirectory()) {
    System.out.println("ERROR: input file " + nomeIn + " is a directory!");
} else if (!fin.canRead()) {
    System.out.println("ERROR: cannot read from input file " + nomeIn + "!");
} else {
    System.out.println("Ficheiro válido!: " + nomeIn);
    System.out.println("Comprimento Ficheiro = " + fin.length());
    System.out.println("Caminho do ficheiro = " + fin.getAbsolutePath());
}
....
```



Class File, exemplos de funções

// lista diretório

```
String[] lista = new String[100];  
File fin1 = new File("c:\\");  
lista = fin1.list();  
for (String n : lista) {  
    System.out.printf("%s\n", n);  
}
```

