Aula 08

Ficheiros em `Java`

Manipulação, Escrita e Leitura de Ficheiros de Texto

Programação 1, 2015-2016

v1.3, 17-11-2015

, DETI, Universidade de Aveiro

08.1

Conteúdo

08.2

1 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 se armazenar de uma forma persistente a informação gerada pelos nossos programas, podemos guardá-la 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.

08.3

2 Manipulação de ficheiros e directórios

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.

08.4

Utilização de ficheiros em Java

- Tipo de dados `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.
  - ...
- Documentação: `view-javadoc File`
- Exemplo: comando `ls`

08.5

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`, ...)
- Comando (`linux`): `iconv`
- Comandos (`linux`, prog2): `latin1-2-utf8`, `utf8-2-latin1`

08.6

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 serviços de entrada/saída fazem automaticamente a tradução de ou para a codificação escolhida;
- Existem constantes literais para expressar caracteres latin1 (`'\xxx'`) ou unicode (`'\uxxxx'`);
- Existem também constantes literais para alguns caracteres especiais:
  - `'\n'`: nova linha;
  - `'\t'`: tabulação horizontal;
  - `'\"'`: carácter `"`
  - `'\''`: carácter `'`
  - `'\\'`: carácter `\`

08.7

3 Escrita de ficheiros de texto em Java

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

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

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

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

3. Escrever sobre o ficheiro:

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

4. Fechar o ficheiro

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

08.8

Exemplo: escreve entradas para ficheiro

```
1. Ler nome do ficheiro
2. Abrir ficheiro para escrita
3. repetir
   3.1. Ler linha de texto
   3.2. Se linha não vazia então
      3.2.1. Escreve linha no ficheiro
4. Fecha ficheiro
```

08.9

Alternativa para escrita de ficheiros de texto

- Tipo de dados `FileWriter` (`java.io.FileWriter`);
- Podemos utilizar objectos `FileWriter` na criação do `File` ou em sua substituição na criação do `PrintWriter`.
- A vantagem é permitir a escrita sucessiva de nova informação sem necessariamente ter de se apagar a informação anterior:

```
static void diario(String filename) {
  FileWriter f = new FileWriter(filename, true); // append!
  PrintWriter pw = new PrintWriter(f);
  String line;
  do {
    line = sc.nextLine();
    if (!line.isEmpty()) {
      pw.println(line);
    }
  } while(!line.isEmpty());
  pw.close();
}
```

08.10

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;
- O módulo `PrintWriter` da bilblioteca `Java` obrigam o programador a lidar explicitamente com a excepção: `IOException`.
- Nos operações de abertura de ficheiros (não só no módulo `PrintWriter`, mas também no módulo a utilizar para leitura de ficheiros) é necessário lidar explicitamente com este tipo de excepções.

08.11

4 Leitura de ficheiros de texto em `Java`

- Tipo de dados `Scanner` (`java.util.Scanner`);
- Em vez do `System.in` associar o `Scanner` ao ficheiro a ler;
- Utilização:
  1. Criar uma entidade (objecto) `File` associada ao nome do ficheiro desejado:

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

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

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

  3. Ler do ficheiro:

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

  4. Fechar o ficheiro:

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

08.12

Exemplo: comando `cat`

```
1. Ler nome do ficheiro
2. Abrir ficheiro para leitura
3. Enquanto não chegar ao fim do ficheiro
   3.1. Ler linha do ficheiro
   3.2. Escrever linha na consola
4. Fecha ficheiro
```

08.13

Exemplo: comando `grep`

```
1. Ler nome dos ficheiros de entrada e saída
2. Ler texto do padrão
3. Abrir ficheiro para leitura: scf
4. Abrir ficheiro para escrita: pwf
5. Enquanto não chegar ao fim de scf
   5.1. Ler uma linha de texto de scf
   5.2. Se linha verifica padrão então
      5.2.1 Escrever linha em pwf
6. Fechar ficheiros
```

08.14