



IO de Streams e Ficheiros

Adaptado dos slides do livro:
Java, an Introduction to Problem Solving and
Programming, 4^od Walter Savitch

Cátia Vaz



Stream

- Ficheiros podem ser utilizados para armazenar:
 - classes;
 - programas;
 - output de um programa;
 - input para um programa.
- Ficheiros I/O assim como teclado e ecrã I/O são manipulados por ***streams***.



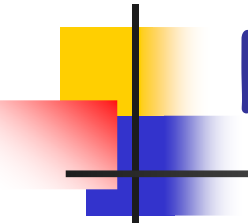
Stream

- Um *stream* é um fluxo de dados (ex: caracteres, números).
- O fluxo de dados para um programa é designado por *input stream*.
- O fluxo de dados de um programa para o exterior é designado por *output stream*.



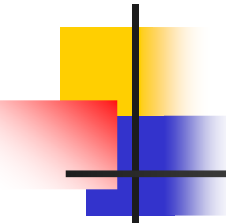
Stream

- Um *stream* é implementado como um objecto.
 - Entrega os dados para um destino (ex:um ficheiro ou outro *stream*) ou
 - Recebe os dados de uma fonte tal como um ficheiro ou o teclado e entrega-os a uma aplicação
 - Exemplo de *output stream* - `System.out`.
 - Exemplos de streams - objectos da classe `Scanner`.



Ficheiros de texto e ficheiros binários

- Todos os dados num ficheiro são armazenados como dígitos binários
- Designam-se por **ficheiros binários** os ficheiros cujo conteúdo tem de ser tratado como sequências de dígitos binários.
- Designam-se por **ficheiros de texto** os ficheiros com streams e métodos que os fazem parecer como sequência de caracteres.



Ficheiros de texto e ficheiros binários

- Embora os ficheiros binários normalmente difiram de máquina para máquina e de linguagem de programação para linguagem de programação, são **mais eficientes** de processar do que os ficheiros de texto.
- Em Java, os ficheiros binários são independentes da plataforma!
 - Mais portabilidade
 - Mais eficiência



Ficheiros de texto e ficheiros binários

- Os ficheiros de texto podem ser escritos e lidos por um editor
- Os ficheiros binários têm de ser lidos e escritos por um programa.



I/O de ficheiros de texto

- **Output:**

- `PrintWriter`
- `FileOutputStream`

- **Input:**

- `BufferedReader`
- `FileReader`
- `StringTokenizer`



Output de ficheiros de texto com PrintWriter

- Um ficheiro pode ser **aberto** do seguinte modo:

```
outputStream = new PrintWriter(  
    new FileOutputStream("out.txt"));
```

- Um ficheiro vazio é conectado a um *stream*.
- Se o ficheiro existir, perde-se o seu conteúdo.
- Se o ficheiro não existir, um novo ficheiro é criado.



Output de ficheiros de texto com PrintWriter

- Pode utilizar-se a classe `FileOutputStream` para criar um *stream* e pode ser utilizado como argumento a um construtor de `PrintWriter`.

- **Sintaxe:**

```
PrintWriter pw = new  
    PrintWriter (new  
        FileOutputStream(NomeFicheiro));
```

ou

```
PrintWriter pw = new  
    PrintWriter (NomeFicheiro);
```



Output de ficheiros de texto com PrintWriter

- O construtor da classe `FileOutputStream` e o da classe `PrintWriter` podem lançar uma excepção do tipo **`FileNotFoundException`**, o que significa que o ficheiro não pode ser criado.
- para criar um *stream*

```
new PrintWriter(new  
    FileOutputStream(NomeFicheiro))
```


ou

```
new PrintWriter (NomeFicheiro);
```
- para adicionar novo texto

```
new PrintWriter(new  
    FileOutputStream(NomeFicheiro, true))
```



Output de ficheiros de texto com PrintWriter

```
public final void println(boolean b)  
public final void println(char c)  
public final void println(char[] c)  
public final void println(double d)  
public final void println(float f)  
public final void println(Object f)  
public final void println(String f)  
/*para escrever como output para o ficheiro  
conectado ao stream*/
```



Output de ficheiros de texto com PrintWriter

```
public final void print(boolean b)  
public final void print(char c)  
public final void print(char[] c)  
public final void print(double d)  
public final void print(float f)  
public final void print(Object f)  
public final void print(String f)  
/*para escrever como output para o ficheiro  
conectado ao stream*/
```



Output de ficheiros de texto com PrintWriter

- `public void close()`

*/*Para fechar a conexão de um stream a um
ficheiro*/*

- `public void flush()`

Quando um programa termina de ler a
partir de um ficheiro ou escrever para
um ficheiro, deve fechar o ficheiro



Input de ficheiros de texto com `BufferedReader`

- A classe `BufferedReader` não tem nenhum construtor cujo argumento seja o nome de um ficheiro.
 - O construtor aceita como argumento um objecto do tipo `Reader`.
 - A Classe `FileReader` tem um construtor que aceita como argumento um nome de um ficheiro e produz um *stream* que é um objecto do tipo `Reader`.



Input de ficheiros de texto com `BufferedReader`

- **Sintaxe**

```
BufferedReader br = new  
    BufferedReader(new  
        FileReader(NomeFicheiro));
```

- O construtor de `FileReader` pode lançar uma exceção do tipo **`FileNotFoundException`**.



Input de ficheiros de texto com `BufferedReader`

- Para criar um *input stream*

```
new BufferedReader(new  
    FileReader(NomeFicheiro))
```

Métodos da classe `BufferedReader`

- `public int read() throws IOException`

*/** para ler um único carácter a partir do
ficheiro, retornando-o como um valor inteiro.
Se a operação de leitura passar o final do
ficheiro, o método retorna `-1`**/*



Métodos da classe BufferedReader

- `public String readLine()
throws IOException`

*/*para ler uma linha a partir do ficheiro.
Se a operação de leitura passar o final
do ficheiro, o método retorna null.*/*



Métodos da classe BufferedReader

- `public void close()`

*/*para fechar uma conexão de um stream a um ficheiro*/*

- Para ler um número com mais de um dígito a partir de um ficheiro de texto, o número tem que ser lido como uma `String` e a `String` tem de ser convertida para um número.

Exemplo-FileCopy

```
public static void fileCopy(String fileNameIn, String fileNameOut) throws
    IOException, FileNotFoundException{
    BufferedReader fileIn=null;
    PrintWriter fileOut=null;
    try{
        fileIn = new BufferedReader(new FileReader(fileNameIn));
        fileOut = new PrintWriter(fileNameOut); int nrLines = 0; String line;
        while((line=fileIn.readLine()) != null){
            fileOut.print (line); fileOut.print("\r\n");\\fileOut.println(line);
            nrLines++;
        }
        System.out.println("Foram copiadas " + nrLines + " linhas");
    }
    finally{
        if(fileOut!=null) fileOut.close();
        if(fileIn!=null) fileIn.close();
    }
}
```

Código de liberação de recursos num bloco finally é sempre uma boa prática, embora nem sempre seja necessário.

cada vez



Exemplo-FileCopy

```
import java.io.FileReader; import java.io.PrintWriter;
import java.io.BufferedReader; import java.io.IOException;
import java.io.FileNotFoundException;
public class FileCopy{

    public static void fileCopy(String fileNameIn, String fileNameOut) throws
        IOException, FileNotFoundException{

        //...
    }

    public static void main(String [] args)throws IOException, FileNotFoundException{
        if(args.length < 2){
            System.out.println("Numero de parametros errado!!!");
            System.exit(0);
        }
        fileCopy(args[0],args[1]);
    }
}
```



Usar a classe File

- Os métodos da classe **File** podem verificar propriedades dos ficheiros.
 - Será que existe o nome do ficheiro?
 - Será que o ficheiro pode ser lido?



Métodos da classe File

```
public boolean exists()  
public boolean canRead()  
public boolean canWrite()  
public boolean delete()  
public long length()  
public String getName()  
public String getPath()  
public boolean isDirectory()  
public File[] listFiles()  
...
```



Exemplo

```
public static boolean chavetas(String fileNameIn) throws IOException,
                                FileNotFoundException{
    BufferedReader fileIn=new BufferedReader(new FileReader(fileNameIn));
    int nChavetas = 0;
    int ler;
    while((ler=fileIn.read()) != -1){
        if( ler=='{') nChavetas++;
        if( ler=='}') {nChavetas--; if(nChavetas<0) return false;}
    }
    fileIn.close();
    return ((nChavetas!=0)? false:true);
}
```

Código de libertação de recursos num bloco finally é sempre uma boa prática, embora nem sempre seja necessário.

Cátia Vaz



Classe StringTokenizer

- A classe **StringTokenizer** pode receber uma linha de texto e dividi-la em palavras individuais.
- A classe **StringTokenizer** está no *package* `java.util`.
- Uma palavra individual é denominada *token*.
- *Tokens* são sequências de caracteres diferentes de espaços.

- **Exemplo**

```
StringTokenizer tokenizer =  
    new StringTokenizer("Ola Mundo!");  
while (tokenizer.hasMoreTokens())  
    System.out.println(tokenizer.nextToken());
```

- **Produz**

```
Ola  
mundo!
```



Classe StringTokenizer

- Separadores são caracteres espaço a menos que outros sejam especificados. Para especificar o conjunto de caracteres separadores, na construção deve ser dado como segundo argumento uma string com todos os caracteres separadores.

- **Exemplo**

```
... new StringTokenizer("Ola Mundo", " ");
```

O delimitador é apenas o caracter ' '

O delimitadores são os caracteres espaço em branco

→

```
... new StringTokenizer("Ola Mundo", " \t\n\r\f");
```

```
...new StringTokenizer("Ola Mundo", " \t\n\r\f.");
```

O delimitadores são os caracteres espaço em branco e o caracter '.'



Métodos da classe StringTokenizer

- Construtores

```
public StringTokenizer(String theString)
public StringTokenizer(String theString,
                        String
                        delimiters)
```

- Verifica se existem mais tokens?

```
public boolean hasMoreTokens()
```

- Obter o próximo token

```
public String nextToken()
```

- Obter o número de tokens que faltam obter

```
public int countTokens()
```

Método split

- Separa a String, usando como separador uma expressão regular

```
public String[] split(String regex)
```

- **Exemplo** Seja `s="Ola Mundo !"`

Dois caracteres ' '.

Um caracter '\t'.

```
String[] x=s.split(" "); x[0]="Ola"; x[1]="Mundo !"
```

' '

'\t'

```
String[] x=s.split(" +"); x[0]="Ola"; x[1]="Mundo !"
```

'\t'

Método split

- Separa a String, usando como separador uma expressão regular

```
public String[] split(String regex)
```

- **Exemplo** Seja `s="Ola Mundo!"`

Dois caracteres ' \ '.

Um caracter '\t'.

```
String[] x=s.split("\\s")
```

`x[0]="Ola"; x[1]=" Mundo!"`

' \ '

```
String[] x=s.split("\\s+");
```

`x[0]="Ola"; x[1]="Mundo!"`

Nota: ao usar a expressão regular `\\s+` e invocando o método para uma linha vazia, é retornado um array de dimensão 1 com a linha vazia.