8 – Ficheiros

8.1 - A classe **File**

Antes de podermos ler/escrever de/para um ficheiro temos que criar um objecto do tipo **File** e associar esse objecto ao ficheiro a que queremos aceder.

Para isso usamos um dos construtores da classe java.io.File:

Por exemplo,

```
File f1= new File (“d://My_work/primeiro.txt");
```

associa ao objecto f1, o ficheiro primeiro.txt, caso exista.

8 – Ficheiros

Podemos verificar se um objecto do tipo File está associado a um ficheiro existente através do método **boolean exists()**

Um objecto do tipo File pode estar associado a uma directoria

```
File directoria = new File ("d://_POO/POO_21_22");
```

Para visualizar os ficheiros de uma directoria usamos o método:

```
String [ ] list();
```

```
String ficheiros[ ] = directoria.list();
   for ( int i=0; i < ficheiros.length; i++){
      System.out.println( ficheiros[i]);
   }
```

8 – Ficheiros

Os métodos **boolean isFile()** e **boolean isDirectory()**
permitem verificar se um objecto do tipo File está associado a um ficheiro ou a uma directoria.

```
if ( f1.isFile() )  (System.out.println (“Ficheiro”);

if ( f1.isDirectory() )  (System.out.println (“Directoria”);
```

8 – Ficheiros

8.2 Streams de I/O

Uma “stream” é uma abstracção que representa

uma **fonte** genérica de entrada de dados ou

um **destino** genérico para escrita de dados,

é definida independentemente do dispositivo físico concreto.

Todas as classes que implementam streams de I/O em Java são subclasses das classes abstractas:

8 – Ficheiros

InputStream e **OutputStream** para ler/ escrever bytes

e das classes abstractas

Reader e **Writer** para ler /escrever caracteres (texto).

Subclasses de InputStream e de Reader são fontes de dados

Subclasses de OutputStream e de Writer são destinos de dados

* Uma stream de I/Oé uma sequência de items de dados, geralmente bytes de 8 bits.

8 – Ficheiros

OutputStream

|__ ByteArrayOutputStream

|__**FileOutputStream**

|__FilterOutputStream

| |__BufferedOutputStream

| |__**DataOutputStream**

|__PipedOutputStream

|__**ObjectOutputStream**

InputStream

|__ ByteArrayInputStream

|__**FileInputStream**

|__FilterInputStream

| |__BufferedInputStream

| |__**DataInputStream**

|__PipedInputStream

|__**ObjectInputStream**

8 – Ficheiros

Writer

|__BufferedWriter

| |__LineNumberWriter

|__PrintWriter

|__OutputStreamWriter

| |__**FileWriter**

|__PipedWriter

|__StringWriter

|__CharArrayWriter

Reader

|__BufferedReader

| |__LineNumberReader

|__InputStreamReader

| |__**FileReader**

|__PipedReader

|__StringReader

|__CharArrayReader

8 – Ficheiros

8.3 Input / Output de baixo nível

Para ler ou escrever num ficheiro temos de criar um objecto do tipo “stream” que depois é associado a um objecto do tipo File.

As classes **FileInputStream** e **FileOutputStream** definem objectos do tipo stream que nos permitem ler / escrever sequências de bytes em ficheiros.

8 – Ficheiros

Exemplo: Escrever no ficheiro

```
import java.io.*;
public class Um {
  public static void main(String []args){
   // criar o objecto do tipo File, ficheiro:
    File ficheiro = new File( "teste.dat" );
    try {
// associar um objecto do tipo FileOutputStream a um ficheiro
      FileOutputStream os =  new FileOutputStream ( ficheiro);
```

```
…
     byte [] arrayBytes = { 10, 20, 30, 40, 50, 60, 70, 80 };
  // escrever um array de bytes no ficheiro
     os.write( arrayBytes);
  // depois de concluir a escrita, devemos  fechar a stream *
     os.close();
    }  //try
   // os métodos de acesso a ficheiros lançam excepções do tipo IOException
    catch (IOException e){
      System.out.println (e.getMessage());
    }
 }}
```

- Os dados são previamente escritos em memória principal, num buffer de memória temporário (cache). Quando o buffer fica cheio, são copiados para o disco.
- Se quando o programa termina o buffer não foi fechado, podem perder-se dados que ainda não foram copiados para disco.

```
// para ler o ficheiro

File ficheiro = new File( "teste.dat" );
   // podemos saber o tamanho do ficheiro
   int tamanhoFicheiro = (int)ficheiro.length();
   byte [] arrayBytes = new byte [tamanhoFicheiro];
   try {
    // criamos  um objecto do tipo FileInputStream que
 associamos ao ficheiro
    FileInputStream is = new FileInputStream ( ficheiro );
    // ler os dados
     arrayBytes = is.read ();
   for (int i=0; i<tamanhoFicheiro; i++){
       System.out.println( arrayBytes[i]);
   }  is.close() ...
```

8 – Ficheiros

8.4 Input / Output de alto nível

A classe **DataOutputStream** permite-nos fazer o output de tipos primitivos de dados, convertendo-os em sequências de bytes.

Esta classe fornece um acesso de mais alto nível para aceder a um ficheiro. Não está ligada directamente a um objecto do tipo file mas sim a um objecto do tipo **FileOutputStream**.

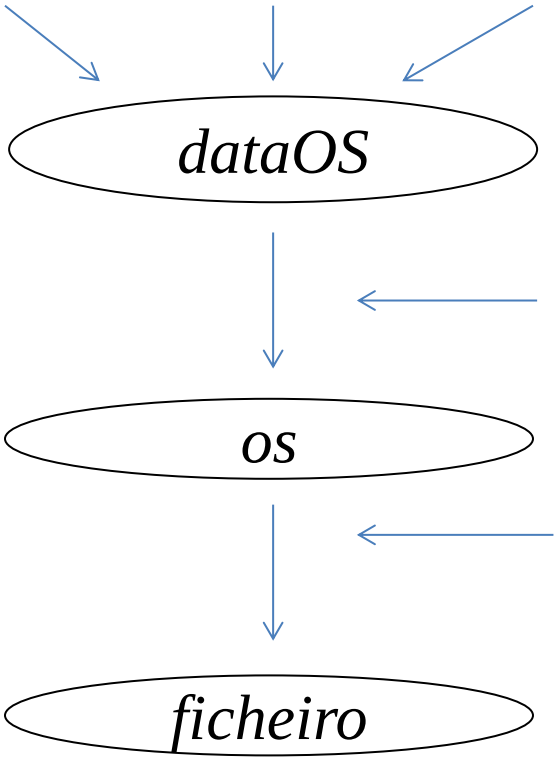
```
File ficheiro = new File("teste2.dat");

FileOutputStream os = new FileOutputStream( ficheiro);

DataOutputStream dataOS = new DataOutputStream(os);
```

8 – Ficheiros

writeFloat writeInt writeDouble <= métodos da classe DataOutputStream



os tipos primitivos são convertidos em sequências de bytes

os bytes são escritos no ficheiro

8 – Ficheiros

Exemplo:

```
File ficheiro = new File("teste2.dat");
 try {
 FileOutputStream os = new FileOutputStream( ficheiro);
 DataOutputStream dataOS = new DataOutputStream(os );
 dataOS.writeInt ( 123456789);
 dataOS.writeDouble(88888888.9);
 dataOS.close();
 }
 catch (IOException e){
   System.out.println(e.getMessage());
```

8 – Ficheiros

Para ler,

revertemos o processo, usando as classes **FileInputStream e DataInputStream**:

Os dados terão que ser lidos pela mesma ordem porque foram escritos.

```
File ficheiro = new File("teste2.dat");

FileInputStream is = new FileInputStream( ficheiro);

DataInputStream dataIS = new DataInputStream( is );
```

```
File ficheiro = new File("teste2.dat");
 try {
 FileInputStream is = new FileInputStream( ficheiro);
 DataInputStream dataIS = new DataInputStream( is );
 System.out.println(dataIS.readInt ());
 System.out.println(dataIS.readDouble());
  dataIS.close();
 }
 catch (IOException e){
   System.out.println(e.getMessage());
 }
```

- a ordem de leitura tem de corresponder à ordem de escrita

8 – Ficheiros

8.5 Ficheiros de Texto

Os dados podem ser armazenados em formato ASCII

Para gerar um ficheiro de texto podemos usar um objecto da classe **PrintWriter**

A classe tem dois métodos de output,

void print (...)

void println (...)

podendo ser o argumento um valor de qualquer um dos tipos primitivos.

8 – Ficheiros

Os métodos convertem o valor do argumento numa String e fazem o seu output. O construtor da classe PrintWriter recebe uma Stream de output.

Exemplo:

```
File ficheiro = new File("teste3.dat");
  try {
  FileOutputStream os = new FileOutputStream( ficheiro);
  PrintWriter pw = new PrintWriter( os );
```

8 – Ficheiros

```
  pw.println ( 123456789); // escrever um int
  pw.println('A'); // um char
  pw.close();
 } ..
```

8 – Ficheiros

Ler um ficheiro de texto

Usamos as classes **FileReader e BufferedReader**

Exemplo:

```
File ficheiro = new File("teste3.dat");
  try {
  FileReader fr = new FileReader( ficheiro);
  BufferedReader br = new BufferedReader(fr);
```

8 – Ficheiros

```
String linha;
 linha = br.readLine();
 int i = Integer.parseInt ( linha);
 linha = br.readLine();
 char c = linha.charAt(0);
 br.close();
 ...
```

8 – Ficheiros

8.6 Input / Output de objectos

Podemos também ler / escrever objectos de / num ficheiro, usando as classes **ObjectInputStream** e **ObjectOutputStream**

Uma **ObjectOutputStream** permite armazenar objectos através do método **writeObject()** que implementa um algoritmo de serialização que garante que todas as referências cruzadas existentes entre instâncias de diferentes classes serão repostas aquando do processo de leitura dessas mesmas instâncias.

8 – Ficheiros

- Para que se possam gravar instâncias de uma determinada classe numa ObjectOutputStream é necessário que a classe **implemente a interface Serializable**.

- Todas as variáveis dessa classe terão que ser também serializáveis. Isto significa que todas as variáveis de instância da classe devem por sua vez pertencer a classes serializáveis.

- Os tipos simples são por definição serializáveis, assim como o são os arrays e as instâncias das classes String e ArrayList.

8 – Ficheiros

Suponhamos a classe C7:

```
public class C7 implements Serializable {
  private int numero;
  private String nome;
  public C7(int n, String nome)    { ...  }
  public void setNumero (int i)   {  ...}
  public void setNome (String n)    {...}
  public int getNumero()    {...}
   public String getNome()     {...}
   public String toString (){
     return (numero+" "+nome + "\n");
  } }
```

8 – Ficheiros

Para gravar objectos:

- criamos uma ObjectOutputStream:

```
File f = new File ("teste4.dat");
FileOutputStream os = new FileOutputStream  (f);
ObjectOutputStream  oOS = new ObjectOutputStream(os);
```

Para escrever um objecto do tipo C7:

- criamos o objecto

```
C7   o1 = new   C7  (1, “XPTO”);
```

- usamos o método **void writeObject(Object)**

```
oOS.writeObject( o1 );
```

Obs. Diferentes tipos de objectos podem ser escritos no mesmo ficheiro.

8 – Ficheiros - Exemplo

```
int i;
C7 objectoC7;
File f = new File ("teste4.dat");
try {
  FileOutputStream os = new FileOutputStream  (f);
  ObjectOutputStream oOS = new ObjectOutputStream(os);
  for (i=0; i<100; i++){
     objectoC7 = new C7( i, "XPTO da Silva");
     oOS.writeObject( objectoC7 );
  }
  oOS.close();
}
catch (IOException e){
  System.out.println(e.getMessage());
}
```

8 – Ficheiros

Para **le**r objectos, usamos as classes **FileInputStream e ObjectInputStream**

…

```
int i;  C7 objectoC7;
File f = new File ("teste4.dat");
try {
  FileInputStream is = new FileInputStream  (f);
  ObjectInputStream oIS = new ObjectInputStream(is);
   for (i=0; i<100; i++){
       // lemos o objecto com o método Object readObject(void);

       objectoC7 = (C7) oIS.readObject();
      // *  temos de converter para o tipo do objecto a ler
          System.out.println ( objectoC7.toString() ) ;
   }
 oIS.close();
} //try
```

8 – Ficheiros

```
catch (IOException e){
        System.out.println(e.getMessage());
 }
catch (ClassNotFoundException e){
        System.out.println("Classe não existente - " +      e.getMessage());
    }
```

Nota: o método **readObject,** além da excepção IOException pode também gerar uma instância de ClassNotFoundException