→ I/O em java (pakage java.io)

→ A classe File

A classe File (subclasse de Object) permite manipular os ficheiros e as directorias de um sistema de ficheiros.

1 - Substituindo o texto *d:\\My_work* pela sua directoria de trabalho, teste o seguinte bloco de código:

```
import java.io.File;
public class c1 {
  public static void main (String args[]){
    File f1= new File ("d:\\My_work\\primeiro.txt");
    if ( f1.exists() )
      System.out.println("O ficheiro existe");
    else
      System.out.println("O ficheiro não existe");
  }
}
```

Como pôde verificar, a criação de um objecto do tipo File não cria o ficheiro em disco.

2 - Explore alguns dos métodos da classe File: delete(); length(); renameTo(File); getName(); e listFiles();

→ Streams

Uma stream é uma abstracção que representa uma fonte genérica de entrada de dados ou um destino genérico para escrita de dados que é definida independentemente do dispositivo físico concreto. Todas as classes que implementam streams em Java são subclasses das classes abstractas

InputStream e **OutputStream** para streams de bytes

e das classes abstractas

Reader e **Writer** para streams de caracteres (texto).

A hierarquia de algumas dessas classes é a seguinte:

OutpuStream
|__ ByteArrayOutputStream
|__**FileOutpuStream**
|__FilterOutputStream
| |__BufferedOutputStream
| |__**DataOutputStream**
|__PipedOutputStream
|__**ObjectOutputStream**

InputStream
|__ ByteArrayInputStream
|__**FileInputStream**
|__FilterInputStream
| |__BufferedInputStream
| |__**DataInputStream**
|__PipedInputStream
|__**ObjectInputStream**

Writer
|__**BufferedWriter**
| |__LineNumberWriter
|__**PrintWriter**
|__OutputStreamWriter
| |__**FileWriter**
|__PipedWriter
|__StringWriter
|__CharArrayWriter

Reader
|__**BufferedReader**
| |__LineNumberReader
|__InputStreamReader
| |__**FileReader**
|__PipedReader
|__StringReader
|__CharArrayReader

→ Streams de caracteres

As subclasses de Writer têm que implementar os métodos definidos nesta classe abstracta, nomeadamente, write(String s), write (int c); write(char[] b); flush(), close(), ...
Analogamente as subclasses de Reader têm que implementar, entre outros, os métodos int read() int read(char[] c); close(), ...

→ As classes FileReader e FileWriter

Construtores: FileReader (File file); FileReader (String filename); ...
FileWriter (File file); FileWriter (String filename); ...

As classes FileReader e FileWriter permitem-nos respectivamente ler e escrever caracteres em objectos do tipo File.

3 – Implemente e estude o exemplo abaixo:

```
import java.io.*;
public class c2 {
  public static void main (String args[]){
   FileWriter fr;
   FileReader fr1;
   String s ="";
   try {
     fr = new FileWriter ( new File ("d:\\My_work\\teste.txt"));
     fr.write("Oi, Boa tarde");
     fr.flush();
     fr.close();
   }
   catch (IOException e){
      System.out.println(e.getMessage());
   }
   try {
     fr1= new FileReader ( new File ("d:\\My_work\\teste.txt"));
     int i = fr1.read();
     while (i != -1){
      s=s+(char)i;
      i=fr1.read();
     }
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
   System.out.println(s );
  }
}
```

4 – Coloque como comentários as instruções: fr.flush(); fr.close(); e tente perceber o que acontece.

→ As classes BufferedReader e BufferedWriter

Construtores: BufferedReader (Reader in)
BufferedWriter (Writer out)

A leitura e a escrita de um carácter de cada vez não é geralmente a forma mais eficiente de manipular ficheiros de texto. As classes BufferedReader e BufferedWriter possuem métodos para leitura e escrita linha a linha.

5 – Teste a classe abaixo e crie uma classe que leia o ficheiro teste2.txt.

```
public class c3 {
  public static void main (String args[]){
    BufferedWriter bw;
    try {
     bw = new BufferedWriter ( new FileWriter ("d:\\My_work\\teste2.txt"));
     bw.write(1);
     bw.newLine();
     bw.write(2);
     bw.flush();
     bw.close();
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
}}
```

A principal vantagem da classe BufferedWriter é que esta realiza escritas optimizadas sobre streams (de caracteres) através de um mecanismo de "buffering". Os dados vão sendo armazenados num buffer intermédio sendo a escrita na stream de destino apenas efectuada quando se atinge o máximo da capacidade do buffer.

Como vimos acima, o construtor da classe BufferedWriter recebe como argumento um objecto da classe Writer, o que significa que uma instância da classe BufferedWriter pode ser definida sobre qualquer subclasse da classe Writer, sempre que for necessário optimizar operações de escrita pouco eficientes.
Simetricamente uma classe BufferedReader pode ser definida sobre qualquer subclasse da classe Reader.

→ A classe PrintWriter

Construtores:
PrintWriter(OutputStream out); PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(Writer out); PrintWriter(Writer out, boolean autoFlush)

As instâncias de PrintWriter podem ser criadas sobre qualquer subclasse de Writer e também sobre uma qualquer stream de bytes (subclasses de OutputStream)
Esta classe define os métodos print() e println() que recebem como parâmetro um valor de **qualquer** tipo simples.

6 – Teste a classe abaixo e seguidamente use a classe PrintWriter sobre uma FileOutputStream. Observe o conteúdo de ambos os ficheiros criados.

```
public class c4 {
  public static void main (String args[]){
    PrintWriter pw;
     try {
     pw = new PrintWriter ( new FileWriter ("d:\\My_work\\teste3.txt"));
     pw.println(2.31);
     pw.println(false);
     pw.print("X");
     pw.flush();
     pw.close();
     }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
}
}
```

→ Streams de bytes

As subclasses de OutputStream e InputStream implementam Streams de bytes. Os métodos abstractos definidos na classe OutputStreams são idênticos aos de Writer com a diferença de que em vez de caracteres aceitam bytes.

→ As classes DataOutputStream e DataInputStream

Construtores: DataOutputStream(OutputStream out);
DataInputStream(InputStream in)

7 – Teste o código abaixo e seguidamente construa uma classe que leia o ficheiro teste6.txt

```
public static void main (String args[]){
    DataOutputStream os;

    try {
     os = new DataOutputStream ( new FileOutputStream("d:\\My_work\\teste6.txt"));
     os.writeDouble(2.335);
     os.writeInt(33);
     os.writeUTF("XPTO");  //Unicode Text Format
     os.flush();
```

```
     os.close();
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
}
```

→ A interface Serializable

A interface *Serializable* não define qualquer método sendo apenas a especificação da propriedade de que as instâncias de uma classe que implemente essa interface poderão ser gravadas em memória secundária sob a forma de objectos.

8 – Implemente e teste a classe abaixo

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

→ As classes ObjectInputStream e ObjectOutputStream

Construtores: ObjectInputStream(InputStream in)
ObjectOutputStream(OutputStream out)

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.

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

9 – Implemente e teste a classe abaixo.

```
public class teste {
  public static void main (String args[]){
     int i;
     C7[] v= new C7[100];
     for (i=0; i<100; i++){
       v[i] = new C7( i, "XPTO da Silva");
     }
     try {
       ObjectOutputStream os = new ObjectOutputStream
         ( new FileOutputStream ("d:\\testeoo.dat"));
       for (i=0; i<100; i++){
          os.writeObject( v[i]);

       }
       os.flush();
      }
      catch (IOException e){
       System.out.println(e.getMessage());
    }//catch  }//main  } //class teste
```

10 – Construa uma classe que lhe permita ler o ficheiro "testeoo.dat" criado anteriormente.