

Leitura e escrita de dados Ficheiros

UA.DETI.POO

Operações de entrada/saída (I/O)

Entrada

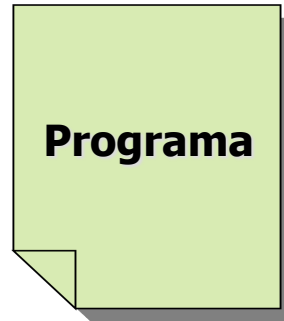
teclado



leitura



Programa



escrita



Saída

monitor



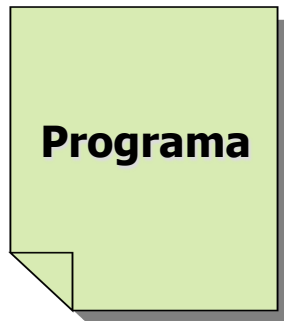
ficheiro



leitura



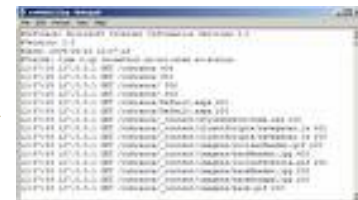
Programa



escrita



ficheiro



Introdução

- ❖ Sem capacidade de interagir com o "resto do mundo", o nosso programa torna-se inútil
 - Esta interação designa-se “input/output” (I/O)
- ❖ Problema → Complexidade
 - Diferentes e complexos dispositivos de I/O (ficheiros, consolas, canais de comunicação, ...)
 - Diferentes formatos de acesso (sequencial, aleatório, binário, caracteres, linha, palavras, ...)
- ❖ Necessidade → Abstração
 - Libertar o programador da necessidade de lidar com as especificidade e complexidade de cada I/O

Java IO e NIO

- ❖ A linguagem java disponibiliza dois packages para permitir operações de entrada/saída de dados

- ❖ **Java IO**

- Stream oriented
- Blocking IO

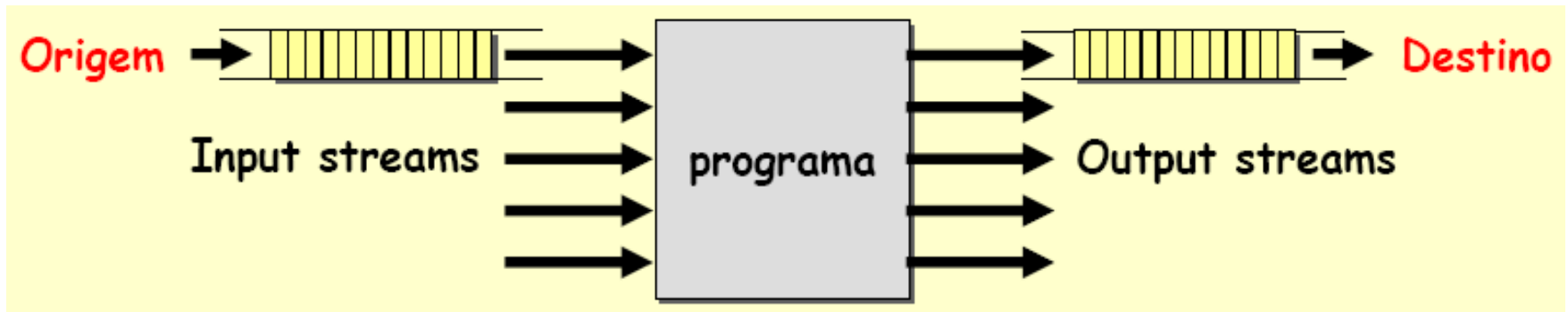
*Vamos discutir
essencialmente java.io*

- ❖ **Java NIO** (new IO)

- Buffer oriented
- Non blocking IO
- Channels
- Selectors

java.io – Stream oriented

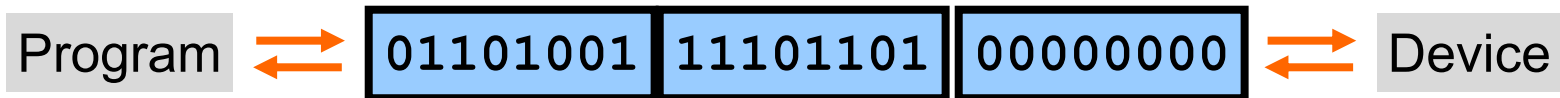
- ❖ O programa pode ler de um *stream* de entrada e escrever num *stream* de saída.
- ❖ Cada stream tem um sentido único



java.io – Tipos de Streams

❖ Byte Streams

- binários (machine-formatted)
- dados transferidos sem serem alterados de forma alguma
- não são interpretados
- não são feitos juízos sobre o seu valor



❖ Character Streams

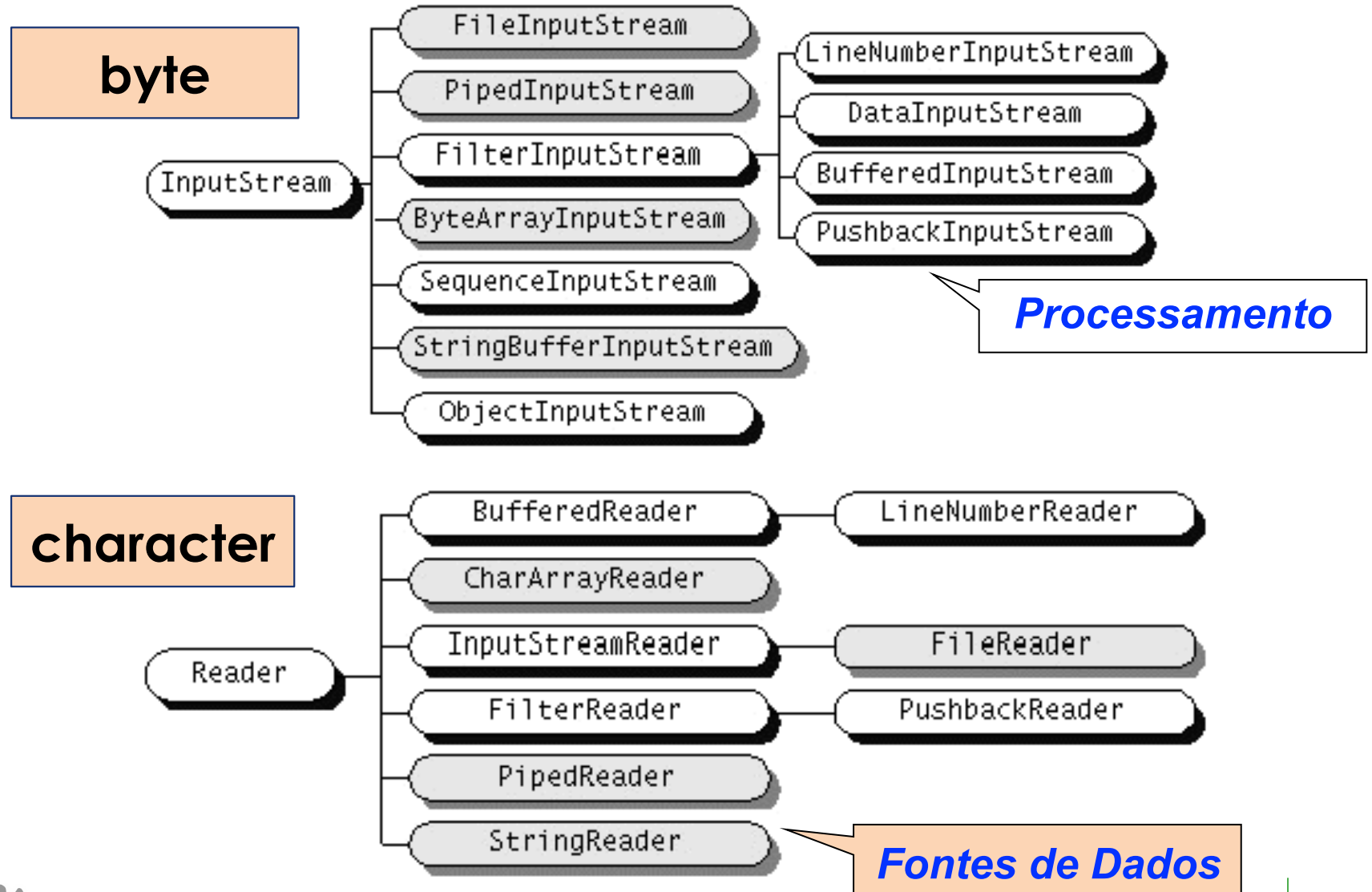
- Os dados estão na forma de caracteres (human-readable data)
- interpretados e transformados de acordo com formatos de representação de texto



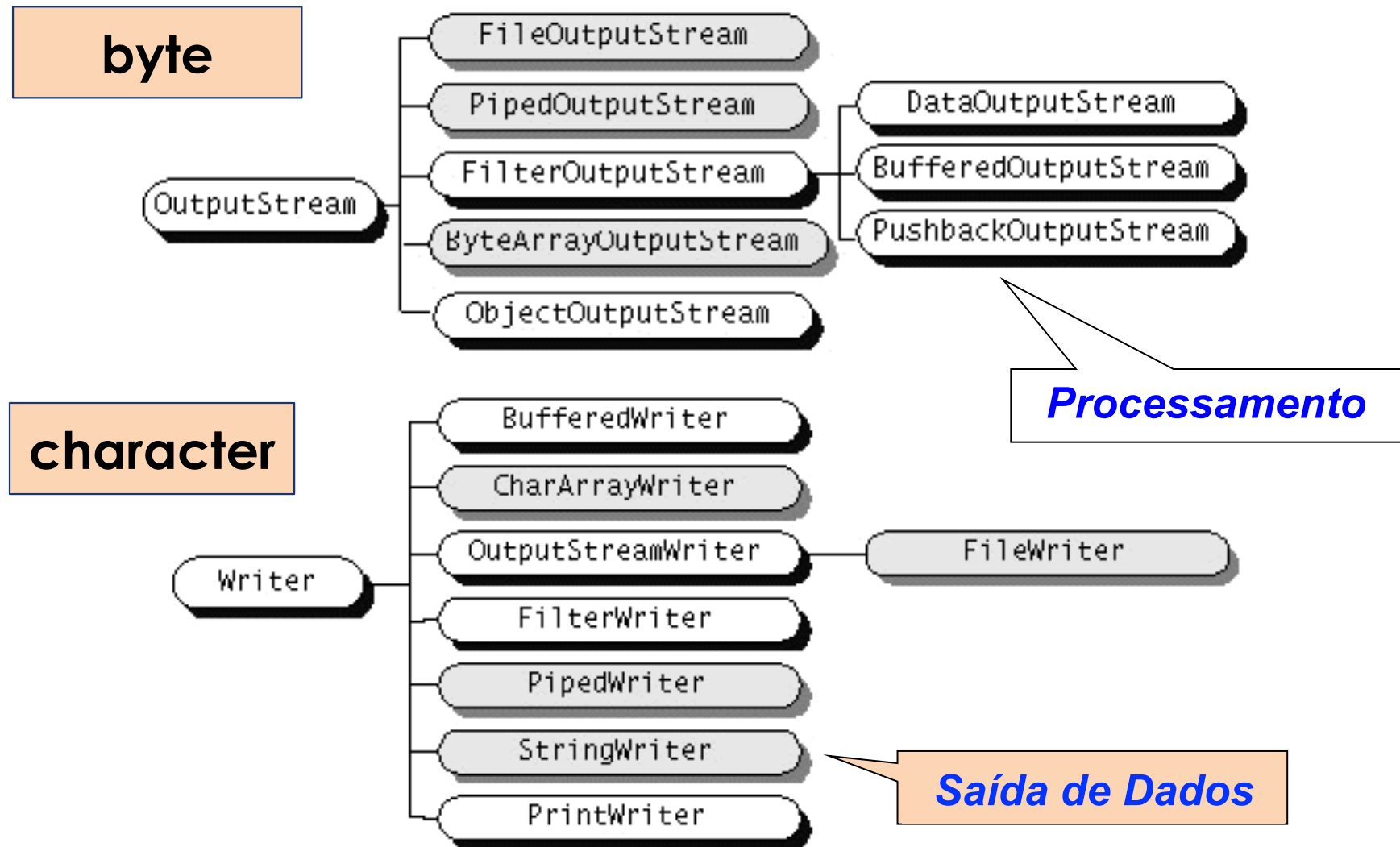
java.io – Streams

- ❖ Temos 4 classes abstratas para lidar com I/O:
 - `InputStream`: byte-input
 - `OutputStream`: byte-output
 - `Reader`: text-input
 - `Writer`: text-output
- ❖ Todas as classes de I/O são derivadas destas
 - Entrada de dados
 - `InputStream (byte)`
 - `Reader (char)`
 - Saída de dados
 - `OutputStream (byte)`
 - `Writer (char)`
- ❖ Estas classes estão incluídas no package **java.io**

Streams de Entrada



Streams de Saída



InputStream/Reader

- ❖ Reader e InputStream têm interfaces semelhantes mas tipos de dados diferentes

- ❖ Reader

```
int read()  
int read(char cbuf[])  
int read(char cbuf[], int offset, int length)
```

- ❖ InputStream

```
int read()  
int read(byte cbuf[])  
int read(byte cbuf[], int offset, int length)
```

OutputStream/Writer

- ❖ Writer e OutputStream têm interfaces semelhantes mas tipos de dados diferentes

- ❖ Writer

```
int write()  
int write(char cbuf[])  
int write(char cbuf[], int offset, int length)
```

- ❖ OutputStream

```
int write()  
int write(byte cbuf[])  
int write(byte cbuf[], int offset, int length)
```

Standard I/O

❖ System.in é do tipo InputStream

```
byte[] b = new byte[10];  
InputStream stdin = System.in;  
stdin.read(b);
```

❖ System.out é do tipo PrintStream (sub-tipo de OutputStream)

```
OutputStream stdout = System.out;  
stdout.write(104); // ASCII 'h'  
stdout.flush();
```

Field Summary		java.lang.System
static PrintStream	err	The "standard" error output stream.
static InputStream	in	The "standard" input stream.
static PrintStream	out	The "standard" output stream.

Ficheiros – Classes principais

❖ Java IO

- File
- Scanner
- FileReader
- FileWriter
- RandomAccessFile

❖ Java NIO

- Path
- Paths
- Files
- SeekableByteChannel

As classes anteriores permitem operações de "baixo nível" sobre streams de dados. Vamos discutir apenas algumas interfaces/classes que permitem operações de "alto nível"

java.io.File

- ❖ A classe *File* representa quer um nome de um ficheiro quer o conjunto de ficheiros num diretório
- ❖ Fornece informações e operações úteis sobre ficheiros e diretórios
 - `canRead`, `canWrite`, `exists`, `getName`, `isDirectory`, `isFile`, `listFiles`, `mkdir`, ...
- ❖ Exemplos:

```
File file1 = new File("io.txt");
File file2 = new File("C:/tmp/", "io.txt");
File file3 = new File("P00/Slides");

if (!file1.exists()) { /* do something */ }
if (!file3.isDirectory()) { /* do something */ }
```

Exemplo – Listar um Diretório

```
import java.io.*;

public class DirList {
    public static void main(String[] args) {
        File directorio = new File("src/");
        File[] arquivos = directorio.listFiles();
        for (File f : arquivos) {
            System.out.println(f.getAbsolutePath());
        }
    }
}
```

Com *java.nio*

Path dir = ...

```
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path entry: stream) { ... }
}
```

java.util.Scanner

- ❖ Classe que facilita a leitura de tipos primitivos e de Strings a partir de um stream de entrada.

- Ler do teclado

```
Scanner sc1 = new Scanner(System.in);  
int i = sc1.nextInt();
```

- Ler de uma string

```
Scanner sc2 = new Scanner("really long\nString\n\t\tthat I want to pick  
apart\n");  
while (sc2.hasNextLine())  
    System.out.println(sc2.nextLine());
```

- Ler de um ficheiro

```
Scanner input = new Scanner(new File("words.txt"));  
while (input.hasNextLine())  
    System.out.println(input.nextLine());
```


Leitura de ficheiros de texto

❖ Exemplo 1: sem tratamento de exceções

```
public class TestReadFile
{
    public static void main(String[] args) throws FileNotFoundException
    {
        Scanner input = new Scanner(new File("words.txt"));
        while (input.hasNextLine())
            System.out.println(input.nextLine());
    }
}
```

❖ O ficheiro "words.txt" deve estar:

- Na pasta local, se o programa for executado através de linha de comando
- Na pasta do projeto, caso seja executado no Eclipse

Leitura de ficheiros de texto

❖ Exemplo 2: try .. catch

```
public static void main(String[] args) {  
    try {  
        Scanner input = new Scanner(new File("words.txt"));  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
        input.close();  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    }  
}
```

– OU

```
public static void main(String[] args) {  
    Scanner input = null;  
    try {  
        input = new Scanner(new File("words.txt"));  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    } finally {  
        if (input != null) input.close();  
    }  
}
```

Leitura de ficheiros de texto

❖ Exemplo 3: try-with-resources

- O código que declara e cria recursos é colocado na entrada try().
- Recursos são objetos que implementam AutoCloseable e que têm de ser fechados depois de usados.

```
public static void main(String[] args) {  
    try ( Scanner input = new Scanner(new File("words.txt"))) {  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    }  
}
```

java.nio – Leitura de ficheiros de texto

❖ Podemos usar métodos estáticos das classes **Files** e **Paths** do package java.nio.file.

❖ Exemplo 4:

```
public class ReadFileIntoList {  
    public static void main(String[] args) throws IOException {  
        List<String> lines = Files.readAllLines(Paths.get("words.txt"));  
        for (String ln : lines)  
            System.out.println(ln);  
    }  
}
```

Escrita de ficheiros de texto

❖ classe java.io.PrintWriter

- Permite-nos usar os métodos println e printf para escrever em ficheiros de texto.
- Formata os valores de tipos primitivos em texto, tal como quando impressos no écran.

```
public class FileWritingDemo {  
    public static void main(String[] args) throws IOException {  
        PrintWriter out = new PrintWriter(new File("file1.txt"));  
        out.println("Fim de semana na praia");  
        out.printf("Viagem: %d\nHotel: %d\n", 345, 1000);  
        out.close();  
    }  
}
```

Escrita de ficheiros de texto – append

- ❖ Podemos acrescentar (append) mais informação a um ficheiro existente

```
public class FileWritingDemo {  
    public static void main(String[] args) throws IOException {  
        FileWriter fileWriter = new FileWriter("file1.txt", true);  
        PrintWriter printWriter = new PrintWriter(fileWriter);  
        printWriter.append("a acrescentar mais umas notas...\n");  
        printWriter.close();  
    }  
}
```

Exercícios

- ❖ Escreva um programa que peça ao utilizador o nome de um ficheiro e que, em seguida, escreva nesse ficheiro todos os números primos de 1 a 1000.
- ❖ Em seguida, escreva um programa que leia o ficheiro criado e que apresente a soma e a média de todos os números.
- ❖ Escreva um programa que repetidamente peça ao utilizador o nome de um ficheiro de texto, até que um que exista. Em seguida, imprima o seu conteúdo em maiúsculas.

Ficheiro binários

❖ `java.io.RandomAccessFile`

- Vê uma file como uma sequência de bytes
- Possui um ponteiro (seek) para ler ou escrever em qualquer ponto do ficheiro.
- Genericamente, inclui operações seek, read, write

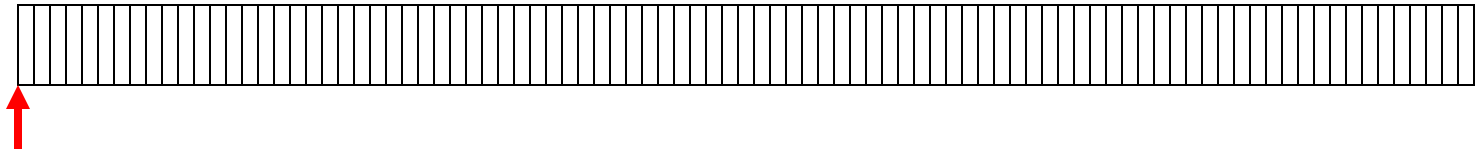
❖ Podemos apenas ler ou escrever tipos primitivos

`writeByte()`, `writeInt()`, `writeBoolean()`
`writeChars(String s)`, `writeUTF(String str)`,
`String readLine()`

Com *java.nio* existem outras classes / métodos
`FileChannel..`

Ficheiro binários – exemplo

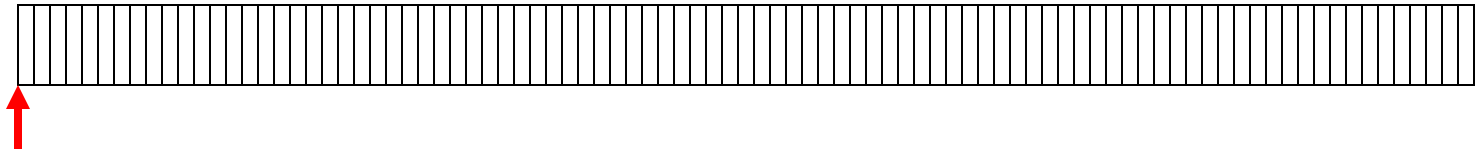
❖ Assumindo a seguinte organização do ficheiro




```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10);  file.read(buf);  
file.seek(0);  file.write(buf);  
file.close();
```

Ficheiro binários – exemplo

❖ Reservar um buffer de 10 bytes

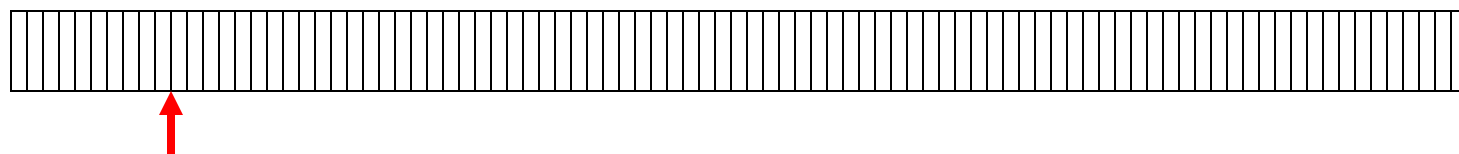


 **buf:** 10 bytes in memory

```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10);  file.read(buf);  
file.seek(0);  file.write(buf);  
file.close();
```

Ficheiro binários – exemplo

❖ Mudar a ponteiro

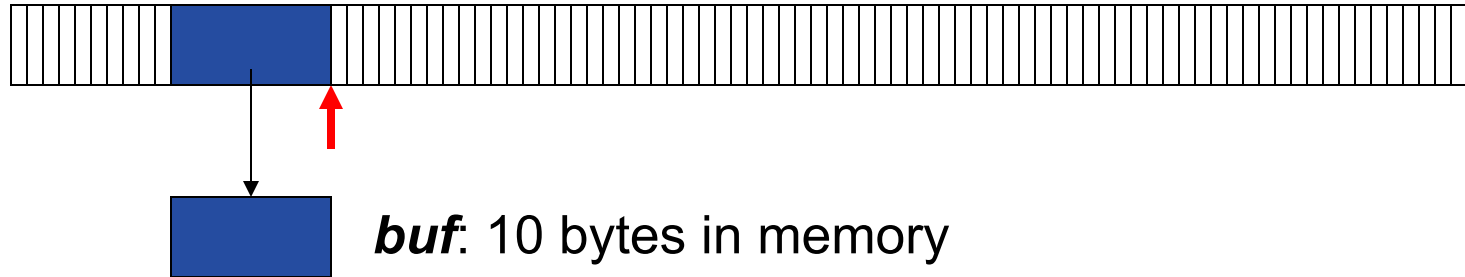


buf: 10 bytes in memory

```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10);  file.read(buf);  
file.seek(0);  file.write(buf);  
file.close();
```

Ficheiro binários – exemplo

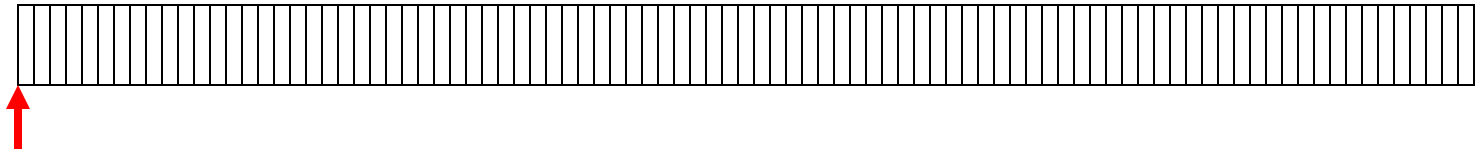
❖ Ler para memória



```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10);  file.read(buf);  
file.seek(0);  file.write(buf);  
file.close();
```

Ficheiro binários – exemplo

❖ Mudar o ponteiro

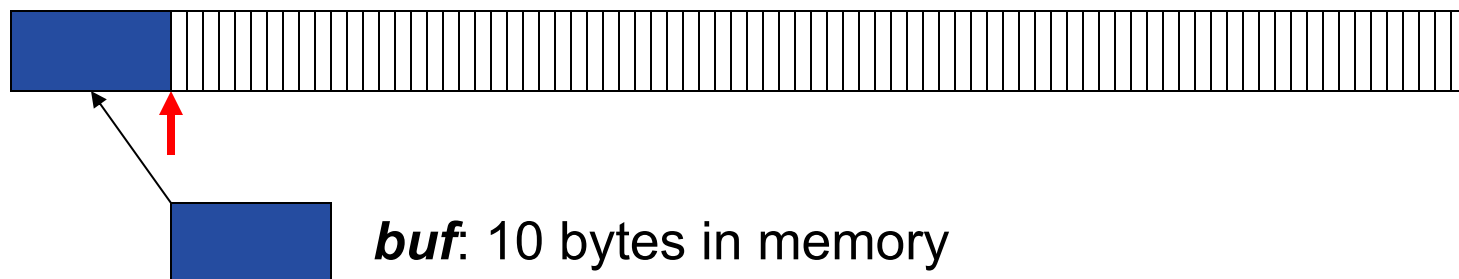


buf: 10 bytes in memory

```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10);  file.read(buf);  
file.seek(0);  file.write(buf);  
file.close();
```

Ficheiro binários – exemplo

❖ Escrever para o ficheiro



```
// In the file “./mydata”, copy bytes 10-19 to 0-9.  
RandomAccessFile file = new RandomAccessFile(“mydata”, “rw”);  
byte[] buf = new byte[10];  
file.seek(10); file.read(buf);  
file.seek(0); file.write(buf);  
file.close();
```

Ficheiro binários

- ❖ Fazer *append* a um ficheiro que já existe.

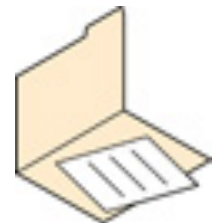
```
File f = new File("um_ficheiro_qualquer");
RandomAccessFile raf = new RandomAccessFile(f, "rw");

// Seek to end of file
raf.seek(f.length());

// Append to the end
raf.writeChars("agora é que é o fim");
raf.close();
```

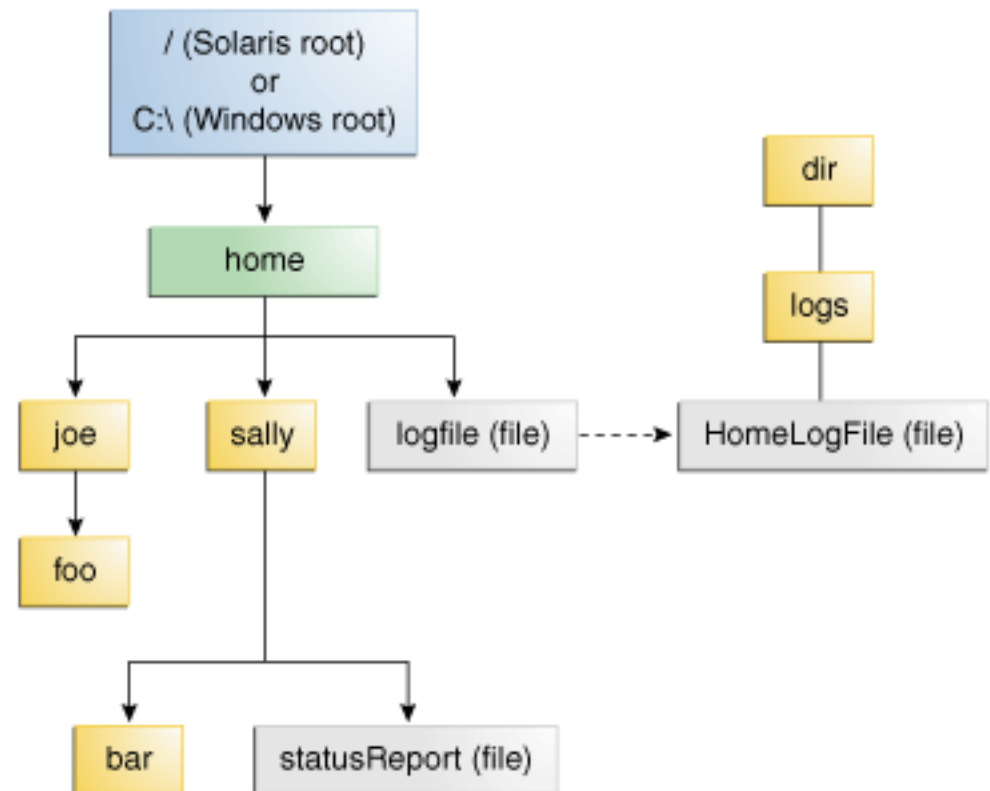
Java NIO (Java 7+)

- ❖ Mudanças significativas nas classes principais
- ❖ Classe **java.nio.file.Files**
 - Só métodos estáticos para manipular ficheiros, directórios,...
- ❖ Classe **java.nio.file.Paths**
 - Só métodos estáticos para retornar um Path através da conversão de uma string ou Uniform Resource Identifier (URI)
- ❖ Interface **java.nio.file.Path**
 - Utilizada para representar a localização de um ficheiro ou sistema de ficheiros, tipicamente system dependent.
- ❖ Utilização comum:
 - Usar Paths para obter um Path.
 - Usar Files para realizar operações.



java.nio.file.Path

- ❖ Notation dependent on the OS
 - /home/sally/statusReport
 - C:\home\sally\statusReport
- ❖ Relative or absolute
- ❖ Symbolic links
- ❖ java.nio.file.Path
 - Interface
 - Path might not exist



java.nio.file.Paths

- ❖ Classe auxiliar com 2 métodos estáticos
- ❖ Permite converter strings ou um URI num Path

`static Path get(String first, String... more)`

- *Converts a path string, or a sequence of strings that when joined form a path string, to a Path.*

`static Path get(URI uri)`

- *Converts the given URI to a Path object.*

java.nio.file.Path

❖ Criar

```
Path p1 = Paths.get("/tmp/foo");  
Path p11 = FileSystems.getDefault().getPath("/tmp/foo");    // <=> p1  
Path p2 = Paths.get(args[0]);  
Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));
```

❖ Criar no home directory logs/foo.log (ou logs\foo.log)

```
Path p5 = Paths.get(System.getProperty("user.home"), "logs", "foo.log");
```

java.nio.file.Path

❖ Alguns métodos:

```
// Microsoft Windows syntax
Path path = Paths.get("C:\\home\\joe\\foo");
// Linux syntax
Path path = Paths.get("/home/joe/foo");

System.out.format("toString: %s\n", path.toString());
System.out.format("getFileName: %s\n", path.getFileName());
System.out.format("getName(0): %s\n", path.getName(0));
System.out.format("getNameCount: %d\n", path.getNameCount());
System.out.format("subpath(0,2): %s\n", path.subpath(0,2));
System.out.format("getParent: %s\n", path.getParent());
System.out.format("getRoot: %s\n", path.getRoot());
```

java.nio.file.Files

❖ Só métodos estáticos

- copy, create, delete, ..
- isDirectory, isReadable, isWritable, ..

❖ Exemplo de cópia de ficheiros

```
Path src = Paths.get("/home/fred/readme.txt");  
Path dst = Paths.get("/home/fred/copy_readme.txt");
```

```
Files.copy(src, dst,  
           StandardCopyOption.COPY_ATTRIBUTES,  
           StandardCopyOption.REPLACE_EXISTING);
```

❖ Move

- Suporta atomic move

```
Path src = Paths.get("/home/fred/readme.txt");  
Path dst = Paths.get("/home/fred/readme.1st");
```

```
Files.move(src, dst, StandardCopyOption.ATOMIC_MOVE);
```

java.nio.file.Files

❖ delete(Path)

```
try {  
    Files.delete(path);  
} catch (NoSuchFileException x) {  
    System.err.format("%s: no such" + " file or directory%n", path);  
} catch (DirectoryNotEmptyException x) {  
    System.err.format("%s not empty%n", path);  
} catch (IOException x) {  
    // File permission problems are caught here.  
    System.err.println(x);  
}
```

❖ deletelfExists(Path)

– Sem exceções

java.nio.file.Files

- ❖ Verificar se dois Paths indicam o mesmo ficheiro
 - Num sistema de ficheiros com links simbólicos podemos ter dois caminhos distintos a representar o mesmo ficheiro
 - Usar `isSameFile(Path, Path)` para fazer a comparação

```
Path p1 = ...;  
Path p2 = ...;  
  
if (Files.isSameFile(p1, p2)) {  
    // Logic when the paths locate the same file  
}
```

Sumário

- ❖ java.io e java.nio
- ❖ Representar ficheiros e directórios com **File**
- ❖ Ler ficheiros de texto com **Scanner**
- ❖ Escrever ficheiros de texto com **PrintWriter**
- ❖ Ler e escrever ficheiros binários com **RandomAccessFile**
- ❖ Muitas outras classes existem para manipular I/O
 - <https://docs.oracle.com/javase/tutorial/essential/io/>