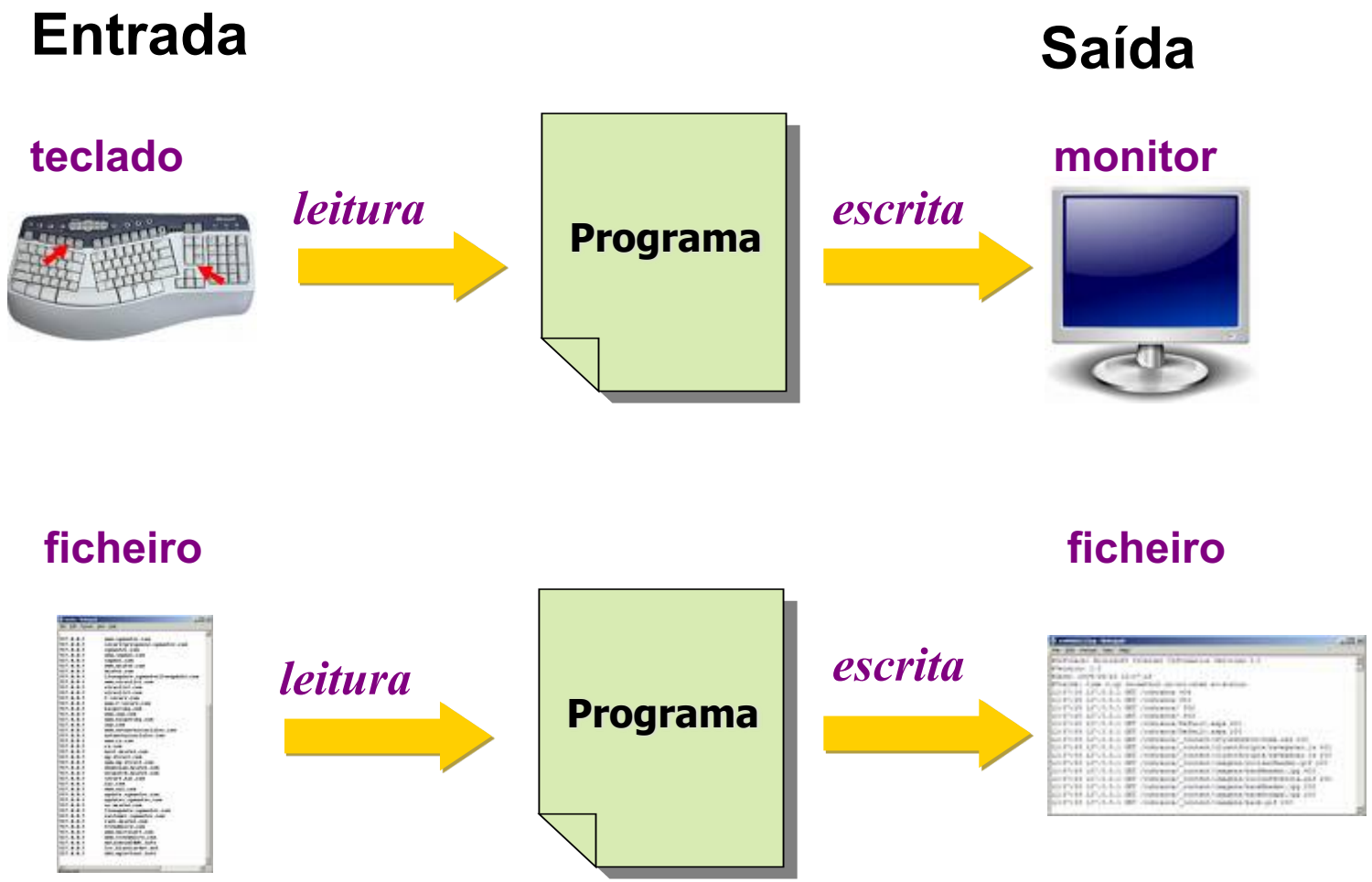
Sistema de Entrada/Saída (I/O)

Operações de entrada/saída

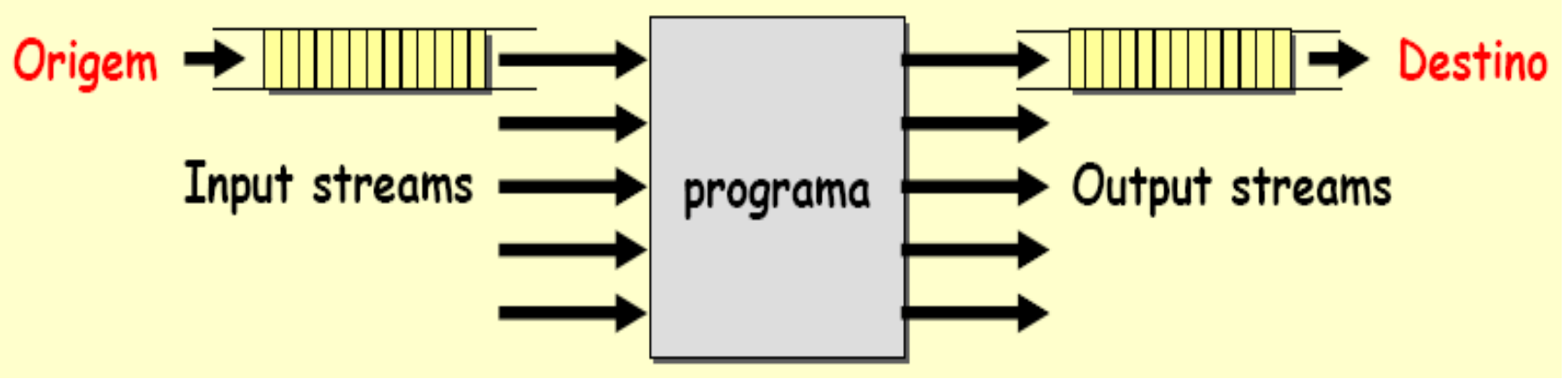


Introdução

- Sem capacidade de interagir com o "resto do mundo", o nosso programa torna-se inútil
  - Esta interacção 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 → Abstracção
  - Libertar o programador da necessidade de lidar com as especificidade e complexidade de cada I/O
- Em Java, a abstracção I/O chama-se “Streams”

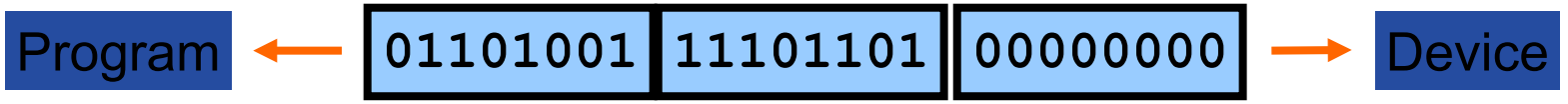
I/O Streams

- O que são Streams?
  - um fluxo de dados que pode entrar ou sair de um programa.

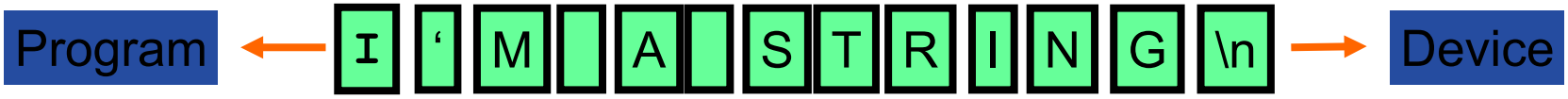


Tipos de Streams

- Byte Streams
  - binárias (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



Streams

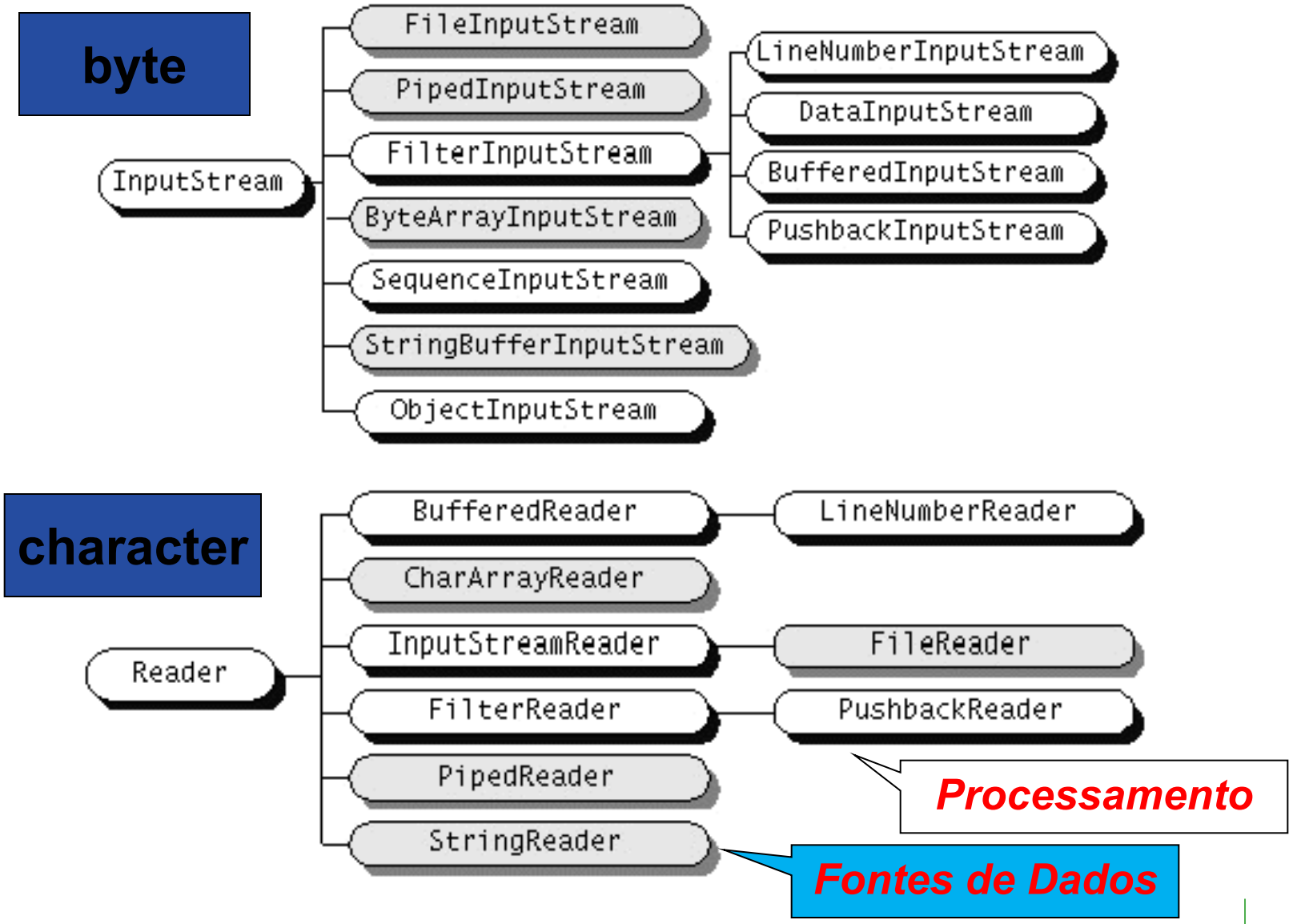
- Os streams são objectos em Java. Temos 4 classes abstractas para lidar com I/O:
  - InputStream: byte-input
  - OutputStream: byte-output
  - Reader: text-input
  - Writer: text-output

Classes Abstractas

- Todas as classes de I/O são derivadas destas
  - Entrada – InputStream (byte); Reader (char)
  - Saída – OutputStream (byte); Writer (char)

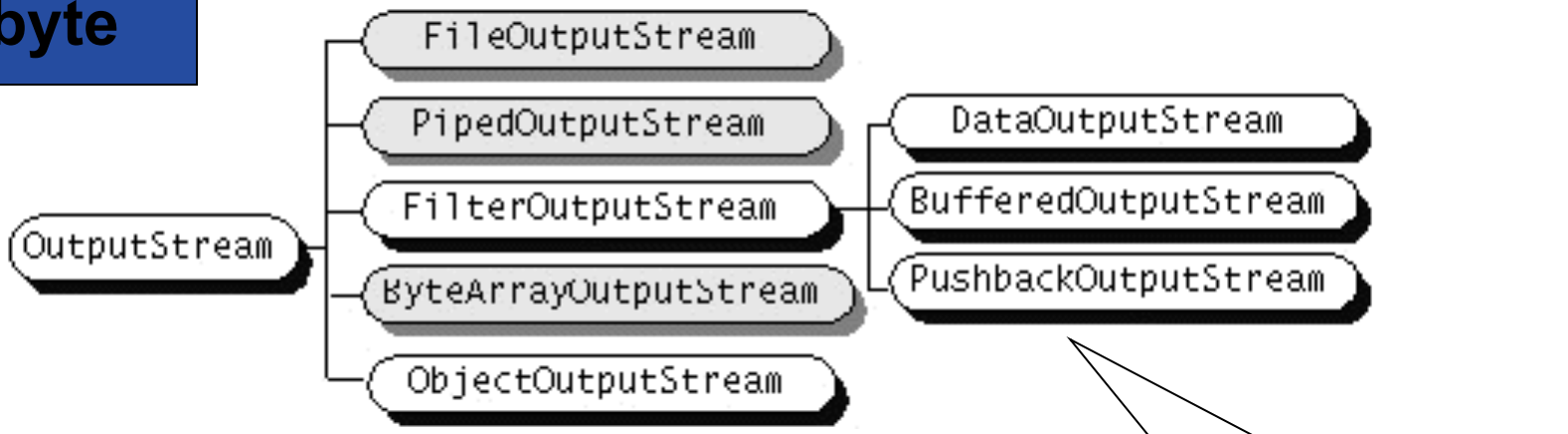
- Estas classes estão incluídas no package java.io

Streams de Entrada

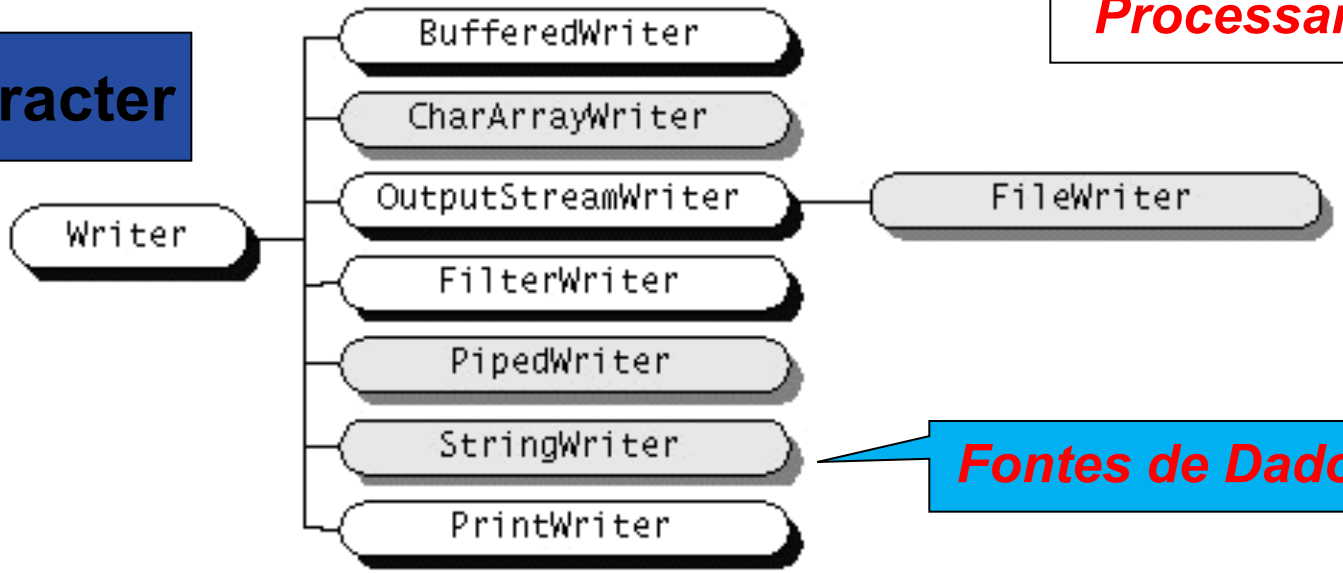


Streams de Saída

- **byte**



character



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.

Utilização de Streams

Sink Type (Fontes de Dados)	**Character Streams**	**Byte Streams**
Memory	CharArrayReader, CharArrayWriter	ByteArrayInputStream, ByteArrayOutputStream
	StringReader, StringWriter	StringBufferInputStream
Pipe	PipedReader, PipedWriter	PipedInputStream, PipedOutputStream
File	FileReader, FileWriter	FileInputStream, FileOutputStream

Classes de processamento

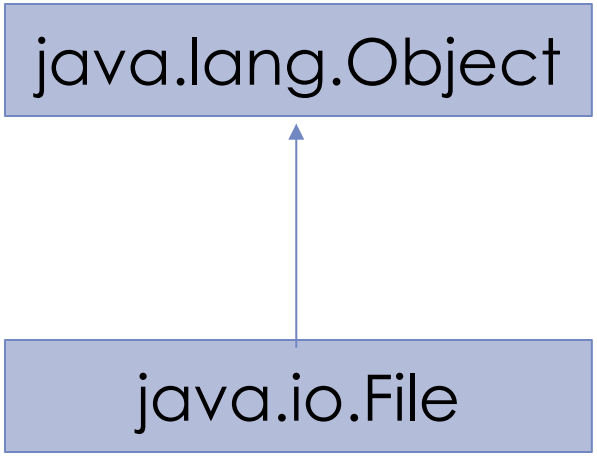
Process	CharacterStreams	Byte Streams
Buffering	BufferedReader, BufferedWriter	BufferedInputStream, BufferedOutputStream
Filtering	FilterReader, FilterWriter	FilterInputStream, FilterOutputStream
Converting between Bytes and Characters	InputStreamReader, OutputStreamWriter	
Concatenation		SequenceInputStream
Object Serialization		ObjectInputStream, ObjectOutputStream
Data Conversion		DataInputStream, DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

Ficheiros

- Classes principais:
  - FileReader
  - FileWriter
  - FileInputStream
  - FileOutputStream
  - File
  - RandomAccessFile
  - Scanner (java 5)
- Java 7
  - Path
  - Paths
  - Files
  - SeekableByteChannel

File (Path no java 7)

- 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, mas não lê ou escreve em arquivos



Exemplo – Criar Directórios e Ficheiros

```
import java.io.*;
public class FileTest {
   public static void main(String[] args)
        throws IOException {
      File directorio = new File("c:/tmp/newdir");
      directorio.mkdirs(); // cria uma árvore

      File arquivo = new File(directorio, "lixo.txt");
      FileOutputStream out =
        new FileOutputStream(arquivo);
      // criar ficheiro
      out.write(new byte[] { 'l', 'i', 'x', 'o' });
      out.close();

      File subdir = new File(directorio , "subdir");
      subdir.mkdir();
      // cria um subdirectório
}
```

A partir de Java 7 existem outros métodos
Files.createFile(..
Files.createDirectory(..

Exemplo – Listar um Directório

```
import java.io.*;

public class DirList {
   public static void main(String[] args)
        throws IOException {
      File directorio = new File("c:/tmp/newdir");
      String[] arquivos = directorio.list();
      for (int i = 0; i < arquivos.length; i++) {
         File filho = new File(directorio, arquivos[i]);
         System.out.println(filho.getAbsolutePath());
      }
}
```

A partir de Java 7..

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

Exemplo - Copiar um Ficheiro Texto

Character Streams

```
import java.io.*;

public class Copy {
    public static void main(String[] args)
        throws IOException {
        File inputFile = new File("input.txt");
        File outputFile = new File("output.txt");

        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;
        while ((c = in.read()) != -1)
           out.write(c);
        in.close();
        out.close();
    }
}
```

Create File Objects

Create File Streams

// Read from Stream

// Write to Stream

Close the Streams

A partir de Java 7 existem outros métodos

FileStreams podem ser criadas sem utilizar um File Object:
FileReader(String fileName)

Exemplo - Copiar um Ficheiro Binário

Byte Streams

```
import java.io.*;

public class Copy {
    public static void main(String[] args) throws IOException {
        File inputFile = new File("picture1.jpg");
        File outputFile = new File("picture2.jpg");

        FileInputStream in = new FileInputStream(inputFile);
        FileOutputStream out = new FileOutputStream(outputFile);
        int c;

        while ((c = in.read()) != -1)
           out.write(c);

        in.close();
        out.close();
    }
}
```

A partir de Java 7 existem outros métodos

Classes de processamento - wrappers

- Exemplos

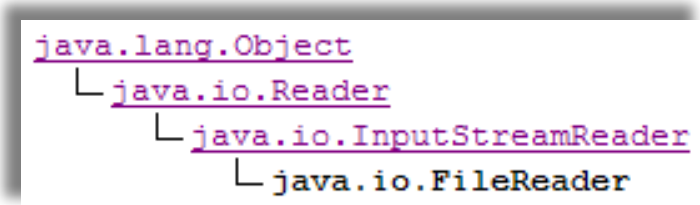
```
DataInputStream src = new DataInputStream(System.in));
BufferedReader in = new BufferedReader(src);
...
PrintWriter out = new PrintWriter(
              new FileWriter("ARQUIVO.TXT"));
```

- A fonte é um objecto do tipo *DataInputStream* que por sua vez é aberto sobre uma outra fonte
  - *DataInputStream* decorado por *BufferedReader*
  - Desta forma pode usar-se o método *readLine* de *BufferedReader*
- Do mesmo modo *FileWriter* é adaptado (wrapped) num *PrintWriter* para que o programa possa usar o método *println*.

BufferedReader

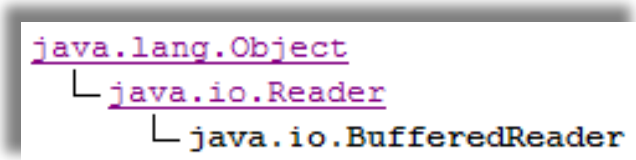
- Leitura de caracteres do System.in

```
InputStreamReader isr = new InputStreamReader(System.in);
char c;
c = (char) isr.read();
System.out.write(c);
```



- Esta leitura caracter-a-caracter não é eficiente!
  - Podemos querer ler uma linha

```
BufferedReader stdin = new BufferedReader(
          new InputStreamReader(System.in));

System.out.print("Digite uma linha:");
String linha = stdin.readLine());
```



Leitura de dados do teclado

- Exemplo de leitura de um inteiro em Java 1.4

```
try {
   BufferedReader r =
      new BufferedReader (
          new InputStreamReader(System.in)) ;
   String s = r.readLine() ;
   int i = (new Integer(s)).intValue() ;
   System.out.println(i) ;
} catch(IOException e) { ... }
```

- .. e em Java 5.0

```
Scanner sc = new Scanner(System.in) ;
int i = sc.nextInt() ;
System.out.println(i) ;
```

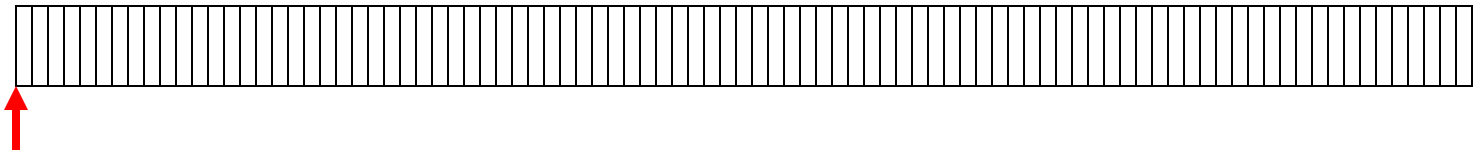
- Compare a manipulação de Excepções!

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()

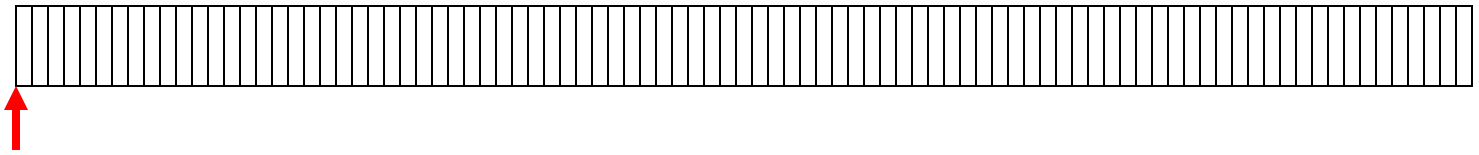
A partir de Java 7 existem outras classes / métodos
... FileChannell

java.io.RandomAccessFile



```
// 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);
```

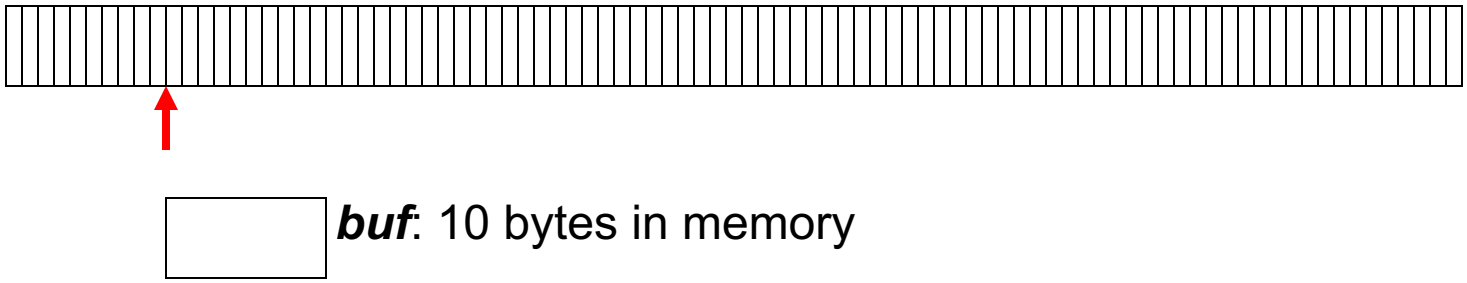
java.io.RandomAccessFile



buf: 10 bytes in memory

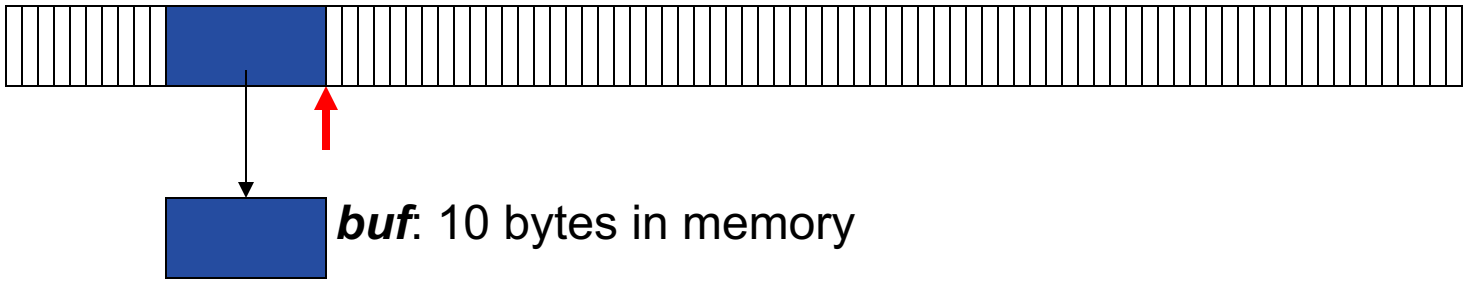
```
// In the file "./mydata", copy bytes 10-19 to 0-9.
RandomAccessFile file = new RandomAccessFile("mydata", "rw");
byte[] buf = new byte[10];
file.seek(10);  file.read(buf);
file.seek(0);  file.write(buf);
```

java.io.RandomAccessFile



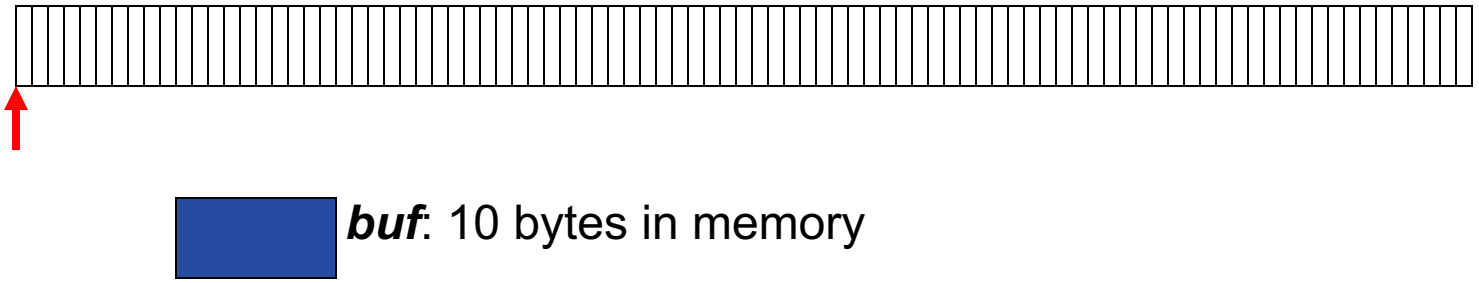

```
// In the file "./mydata", copy bytes 10-19 to 0-9.
RandomAccessFile file = new RandomAccessFile("mydata", "rw");
byte[] buf = new byte[10];
file.seek(10);  file.read(buf);
file.seek(0);  file.write(buf);
```

java.io.RandomAccessFile



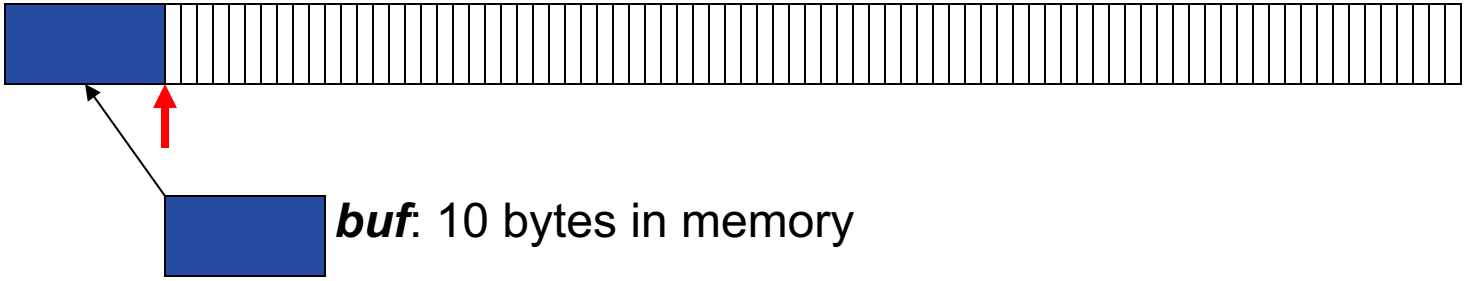

```
// In the file "./mydata", copy bytes 10-19 to 0-9.
RandomAccessFile file = new RandomAccessFile("mydata", "rw");
byte[] buf = new byte[10];
file.seek(10);  file.read(buf);
file.seek(0);  file.write(buf);
```

java.io.RandomAccessFile




```
// 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);
```

java.io.RandomAccessFile




```
// 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);
```

java.io.RandomAccessFile

- Fazer append a um ficheiro que já existe.

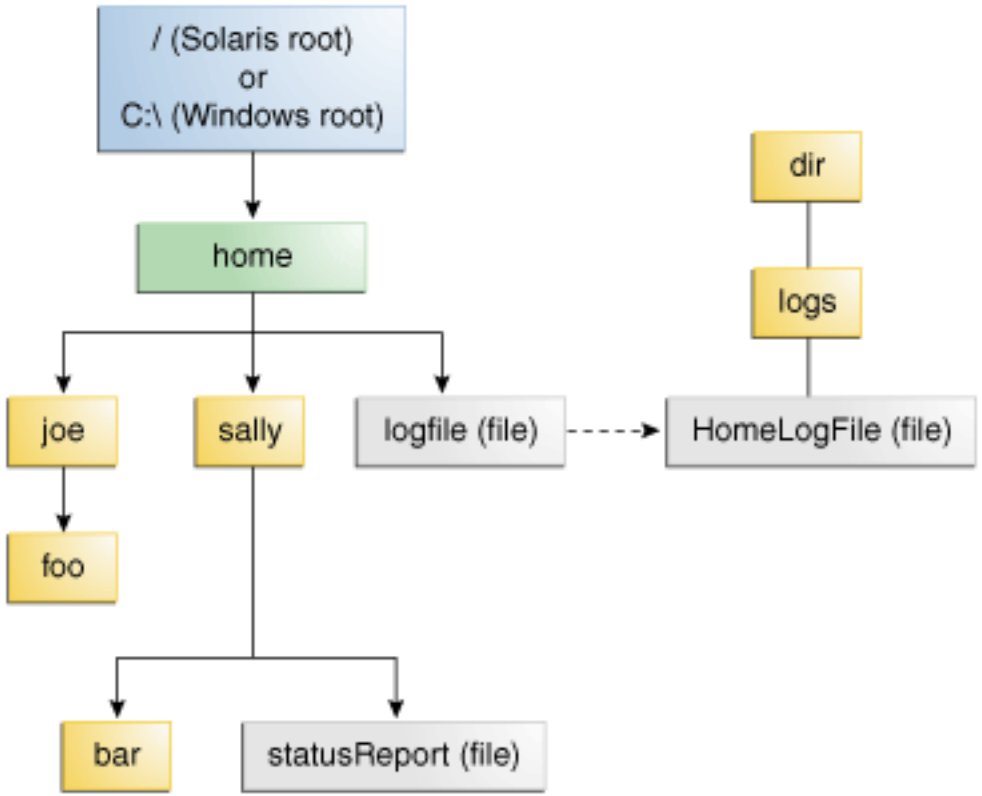
```
File f = new File("um_ficheiro_qualquer");
RandomAccessFile raf = new RandomAccessFile(f, "rw");

// 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.2 (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



http://docs.oracle.com/javase/tutorial/essential/io/pathOps.html

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.*

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");
```

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);
}
```

- deleteIfExists(Path)
  - Sem excepções

java.nio.file.Files

- Verificar se dois Paths indicam a mesma File
  - 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
}
```

java.nio.file.DirectoryStream<T>

- Interface DirectoryStream actua como um iterador
  - *Scales to large directories*
  - *Uses less resources*
  - *Smooth out response time for remote file systems*
  - *Implements Iterable and Closeable for productivity*
- Filtering support
  - *Build-in support for glob, regex and custom filters*

```
Path srcPath = Paths.get("/home/fred/src");
try (DirectoryStream<Path> dir =
      Files.newDirectoryStream(srcPath, "*.java"))  {
  for (Path file : dir)
      System.out.println(file);
 }
```

Resumo de diferenças NIO.2

- http://docs.oracle.com/javase/tutorial/essential/io/legacy.html