

# Fundamentos de Java

Fundamentos de I/O



---

---

---

---

---

---

---

## Tópicos Abordados



- A API de I/O do Java
- Fluxo de dados
  - *InputStream* e *OutputStream*
- Streams em arquivos
- Classes importantes
  - *Scanner*
  - *PrintStream*
  - *File*
- Try-with-resources

---

---

---

---

---

---

---

## A API de I/O do Java



- Está localizada no pacote `java.io`
- A API de I/O gerencia a entrada e saída de dados
  - Console, arquivos, sockets, etc.

---

---

---

---

---

---

---

## Fluxo de Dados



- Todas as operações de I/O são baseadas em fluxo de dados (*streams*)
  - *InputStream*: fluxo de entrada
  - *OutputStream*: fluxo de saída
- A API usa polimorfismo para esconder detalhes de onde a informação vem e para onde ela vai

---

---

---

---

---

---

---

## Fluxo de Dados



- Pelas streams, trafegam bytes
  - *InputStream*: é capaz de ler bytes de algum lugar
  - *OutputStream*: é capaz de escrever bytes em algum lugar

---

---

---

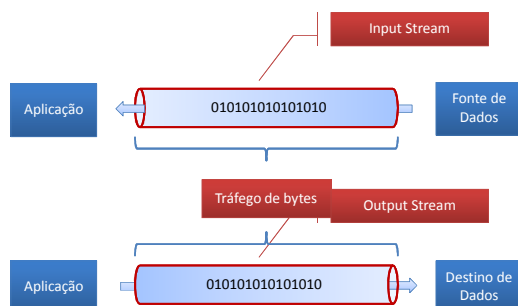
---

---

---

---

## Input Stream e Output Stream



---

---

---

---

---

---

---

## Input Stream




Diagram illustrating Input Stream usage:

**Arquivo (File):** Application reads from a file named "entrada.txt".

```
InputStream is = new FileInputStream("entrada.txt");
int b = is.read();
```

**Console:** Application reads from the system console.

```
InputStream is = System.in;
int b = is.read();
```

---

---

---

---

---

---

---

---

## Output Stream




Diagram illustrating Output Stream usage:

**Arquivo (File):** Application writes to a file named "saida.txt".

```
OutputStream os = new FileOutputStream("saida.txt");
os.write(65);
```

**Console:** Application writes to the system console.

```
OutputStream os = System.out;
os.write(65);
```

---

---

---

---


---

---

---

---

## Lendo Caracteres



- Para lermos caracteres, devemos usar uma "ponte", que converte bytes em caracteres

Diagram illustrating the bridge between `InputStream` and `InputStreamReader`:

`InputStream` (Lê bytes) → `InputStreamReader` (Lê caracteres)

```
InputStream is = new FileInputStream("entrada.txt");
InputStreamReader isr = new InputStreamReader(is);
char c = (char) isr.read();
```

---

---

---

---

---

---

---

---

## Lendo Strings

• Para lermos strings, devemos usar um objeto que consegue juntar os caracteres

```
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
String s = br.readLine();
```

---

---

---

---

---

---

---

---

## Juntando as Classes

---

---

---

---

---

---

---

---

## Escrevendo Caracteres e Strings

```
OutputStream os = new FileOutputStream("saida.txt");
OutputStreamWriter osw = new OutputStreamWriter(os);
BufferedWriter bw = new BufferedWriter(osw);
bw.write("texto");
```

---

---

---

---

---

---

---

---

## Streams em Arquivos



- É possível usar também as classes *FileReader* e *FileWriter* para lermos e escrevermos arquivos texto

```
Reader r = new FileReader("entrada.txt");
Writer w = new FileWriter("saida.txt");
```

Caracteres

```
BufferedReader br = new BufferedReader(r);
BufferedWriter bw = new BufferedWriter(w);
```

Strings

---

---

---

---

---

---

---

## Scanner e PrintStream



- Servem para facilitar o trabalho de ler e escrever dados em streams
  - *Scanner*: lê dados de uma stream de entrada
  - *PrintStream*: escreve dados em uma stream de saída

---

---

---

---

---

---

---

## Scanner



```
Scanner s = new Scanner(new FileInputStream("entrada.txt"));
while(s.hasNextLine()) {
    String token = s.nextLine();
}
```

Possibilidade de trabalhar com tokens

Pode ser utilizado qualquer *InputStream* ou *Reader*

- O *Scanner* possui facilidades para quebrar strings com base em delimitadores

---

---

---


---

---

---

---

PrintStream



```
PrintStream ps = new PrintStream(new FileOutputStream("saida.txt"));
ps.println("texto");
```

Os métodos `print()` e `println()` facilitam a escrita de dados

Pode ser utilizada qualquer `OutputStream`

- `System.out` é uma `PrintStream`

---

---

---

---


---

---

---

---

A Classe `java.io.File`



- Permite acesso às informações sobre um arquivo ou diretório no sistema de arquivos
  - nome, diretório, tamanho em bytes, permissões de escrita e leitura, etc.
- Não representa obrigatoriamente um arquivo existente no sistema de arquivos

---

---

---

---


---

---

---

---

A Classe `java.io.File`



- Como usar

```
File f = new File("C:/Arquivos/arquivo.txt");
```

- Alguns métodos importantes

Método	Descrição
<code>isDirectory()</code>	Informa se é um arquivo ou um diretório
<code>exists()</code>	Informa se o arquivo (ou diretório) existe
<code>getName()</code>	Obtém o nome do arquivo ou diretório
<code>getPath()</code>	Obtém o caminho completo do arquivo ou diretório
<code>listFiles()</code>	Lista os arquivos de um diretório

---

---

---

---


---

---

---


---

Try-with-resources



- Permite o fechamento automático de recursos (chamada ao método `close()`)

```
InputStream is = null;
try {
    is = new FileInputStream("entrada.txt");
    ...
} finally {
    if (is != null) {
        is.close();
    }
}
```



```
try (InputStream is = new FileInputStream("entrada.txt")) {
    ...
}
```

Closeable ou  
AutoCloseable

---

---

---

---

---

---

---

 **Softblue**

---

---

---

---

---

---

---