Acesso ao sistema de arquivos

FURB
UNIVERSIDADE DE BLUMENAU

Categorias das operações de E/S em arquivos

- As operações de E/S em arquivos são divididas em duas categorias:

Operações para acesso ao sistema de arquivos

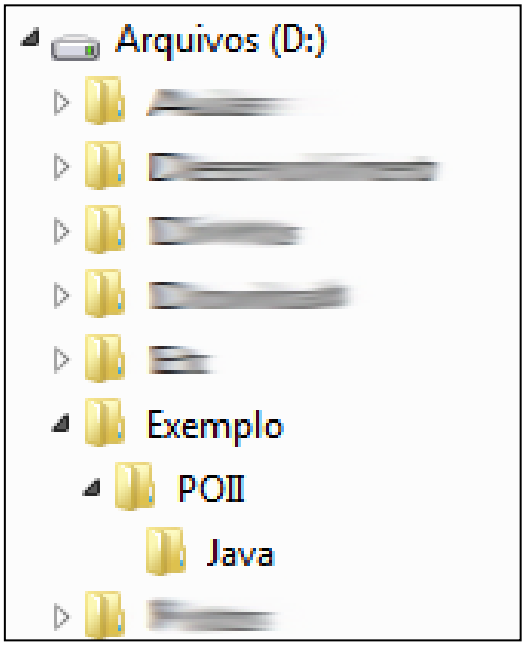
- Obter a relação de arquivos e subdiretórios de um diretório
- Ler propriedades de um arquivo ou diretório
- Criar diretórios, apagar diretórios, renomear arquivos, etc.

Operações de leitura e edição de conteúdo de arquivos

- Ler o conteúdo de arquivos
- Alterar (gravar) dados em arquivos

Diretórios

- **Caminho**: local (diretório) de um arquivo
  - Utiliza-se um caractere separador para expressar o caminho de um arquivo
    - barra (/) ou
    - barra invertida (\)
- **Caminho absoluto**
  - Contém todos os diretórios desde a raiz
    - Exemplo: D:\Exemplo\POII\Java\Slide1.pdf
- **Caminho relativo**
  - O caminho é relativo ao diretório atual
    - Exemplo: Java\Slide1.pdf



Acesso ao sistema de arquivos

A classe **`File`** representa um arquivo ou um diretório.

Membro	Descrição
`File()`	Cria um objeto que representa um arquivo ou um diretório, que pode ou não existir
`getName()`	Retorna o nome do arquivo ou diretório
`exists()`	Retorna true se o arquivo/diretório existe
`isDirectory()`	Retorna true se o objeto representar um diretório
`isFile()`	Retorna true se o objeto representar um arquivo
`length()`	Retorna o tamanho, em bytes, do arquivo (válido apenas para arquivos)
`listFiles()`	Retorna um array com arquivos e diretórios contidos no diretório representado pelo objeto
`createTempFile()`	Cria um arquivo no diretório temporário

Exemplo

```
File diretorio = new File("C:\\Windows");

File[] conteudoDiretorio = diretorio.listFiles();
for (File item : conteudoDiretorio) {
    if (item.isDirectory()) {
        System.out.println("Diretório: " + item.getName());
    } else {
        System.out.println("Arquivo " + item.getName() +
                           "tem " + item.length() + " bytes");
    }
}
```

Persistência de arquivos binários

Introdução

FURB
UNIVERSIDADE DE BLUMENAU

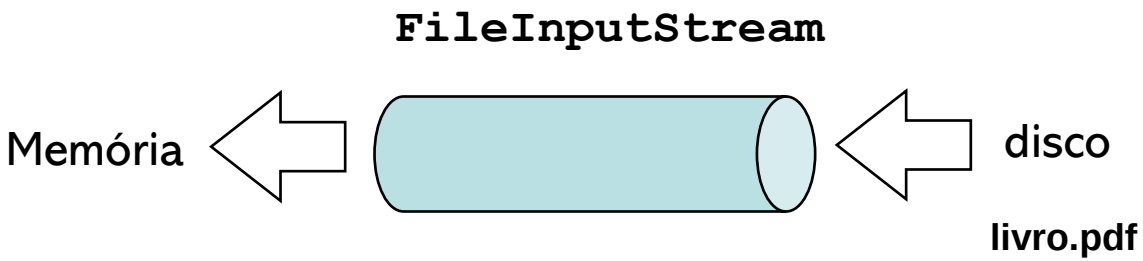
Arquivo Texto X Arquivo Binário

- Arquivo texto
  - Os bits representam caracteres
  - Podem ser lidos por editores de texto
  - São “legíveis” para os humanos

- Arquivos binários
  - Os bits representam dados
  - Utilizam qualquer sequencia de bytes
  - Mais eficiente de processar

Stream

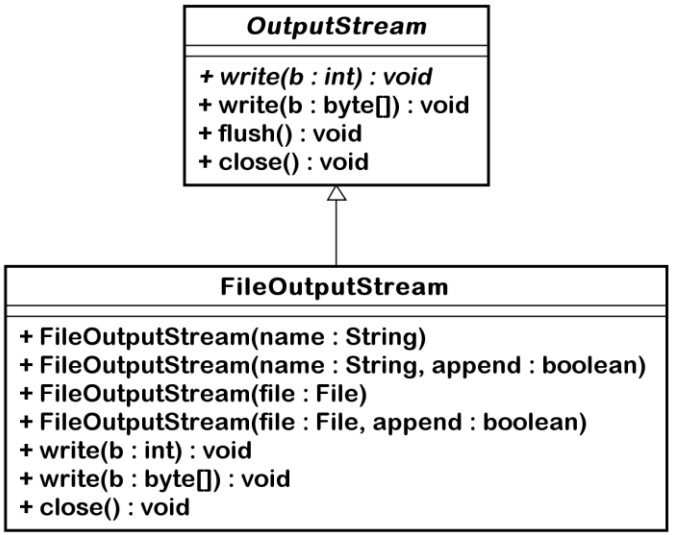
- Um **Stream** é um objeto que tanto obtém dados de uma fonte (como teclado, arquivo, rede) como grava dados (tela, arquivo, etc).



Gravação de arquivos binários

FileOutputStream

A classe **`FileOutputStream`** é utilizada para gravar arquivos.



Membro	Descrição
`FileOutputStream(File)`	Cria um objeto para gravar um arquivo. Se o arquivo já existir, será recriado (destruindo o conteúdo que havia).
`FileOutputStream(File, boolean)`	Cria um objeto para gravar um arquivo, possibilitando acrescentar novos dados no arquivo.
`write(int)`	Grava um byte no arquivo
`write(byte[])`	Grava os dados de um array no arquivo
`flush()`	Provoca a atualiza no arquivo
`close()`	Fecha o arquivo

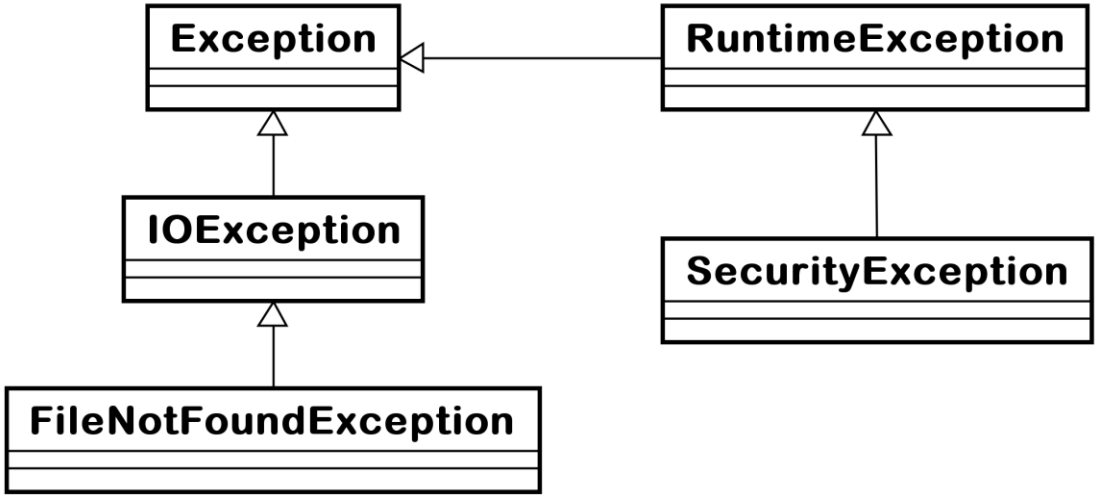
Exemplo

```
File arquivo = new File("D:\\dados.dat");
FileOutputStream fos = new FileOutputStream(arquivo);
fos.write(79);
fos.write(105);
fos.close();
```

O construtor **`FileOutputStream(File,boolean)`** permite indicar que se quer acrescentar dados.

```
File arquivo = new File("D:\\dados.dat");
FileOutputStream fos = new FileOutputStream(arquivo, true);
fos.write(65);
fos.close();
```

Exceções que podem ser lançadas na gravação



Exceção	Descrição
`SecurityException`	Usuário sem permissão para gravar dados no arquivo
`FileNotFoundException`	Diretório não existe
`IOException`	Falha de gravação

Leitura de arquivos binários

FURB
UNIVERSIDADE DE BLUMENAU

Ponteiro do Arquivo

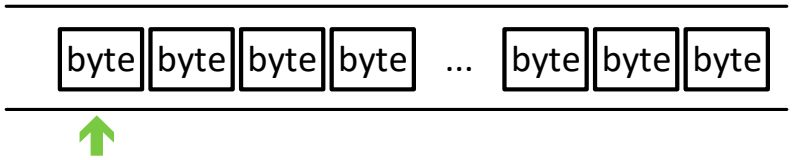
- O ponteiro é um número (`long`) que representa o número do byte do arquivo (começando de 0) em que o ponteiro está posicionado.

- Qualquer operação de leitura ou gravação sempre é feita na posição atual do ponteiro do arquivo.

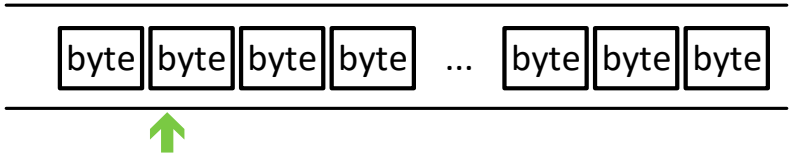
- O ponteiro avança automaticamente quando ocorre uma operação de leitura.

Ponteiro do arquivo

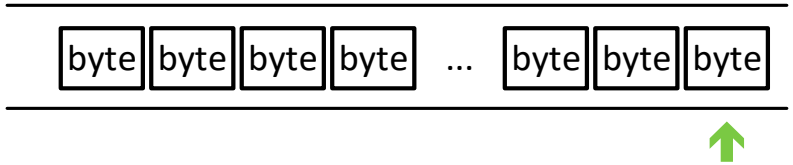
❶ Após abrir o arquivo:



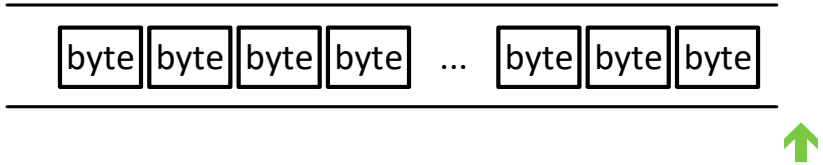
❷ Após executar **`fileInputStream.read()`**:



❸ Depois de vários **`fileInputStream.read()`**...

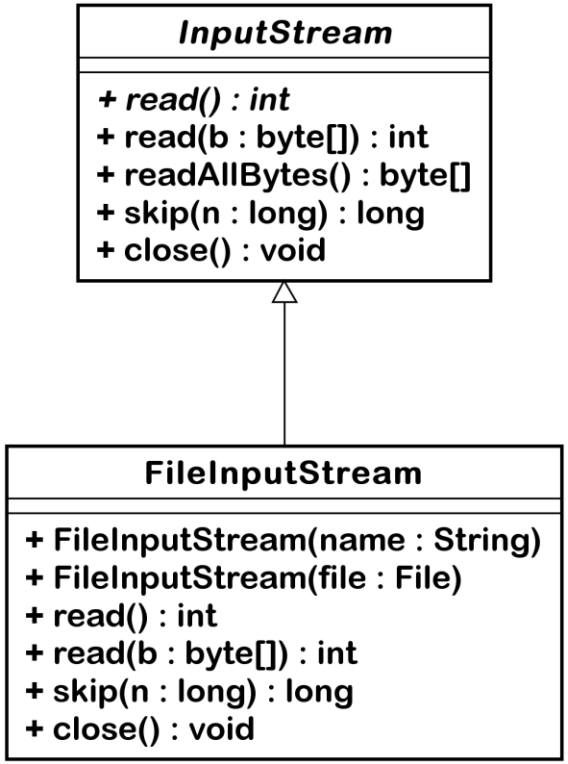


❹ Após executar **`fileInputStream.read()`**:



FileInputStream

A classe **`FileInputStream`** é utilizada para ler arquivos.



Membro	Descrição
`FileInputStream(File)`	Cria um objeto para ler dados de um arquivo.
`read()`	Lê um byte do arquivo. Retorna -1 quando atingir o final do arquivo.
`read(byte[])`	Lê os dados do arquivo, suficiente para alimentar o vetor. Retorna quantidade de bytes lidos.
`readAllBytes()`	Lê e retorna um vetor com todos os dados do arquivo.
`skip()`	Salta para uma posição do arquivo, em relação à posição atual
`close()`	Fecha o arquivo

Exemplo

```
File f = new File("D:\\dados.dat");
FileInputStream fis = new FileInputStream(f);
int dado;

while (true) {
    dado = fis.read();
    if (dado != -1) {
        System.out.println(dado);
    } else {
        break;
    }
}
fis.close();

System.out.println("Terminou");
```

```
File f = new File("D:\\dados.dat");
FileInputStream fis = new FileInputStream(f);
int dado;

while ((dado = fis.read()) != -1) {
    System.out.println(dado);
}

fis.close();
```

Persistência de dados primitivos em arquivos binários

FURB
UNIVERSIDADE DE BLUMENAU

Motivação

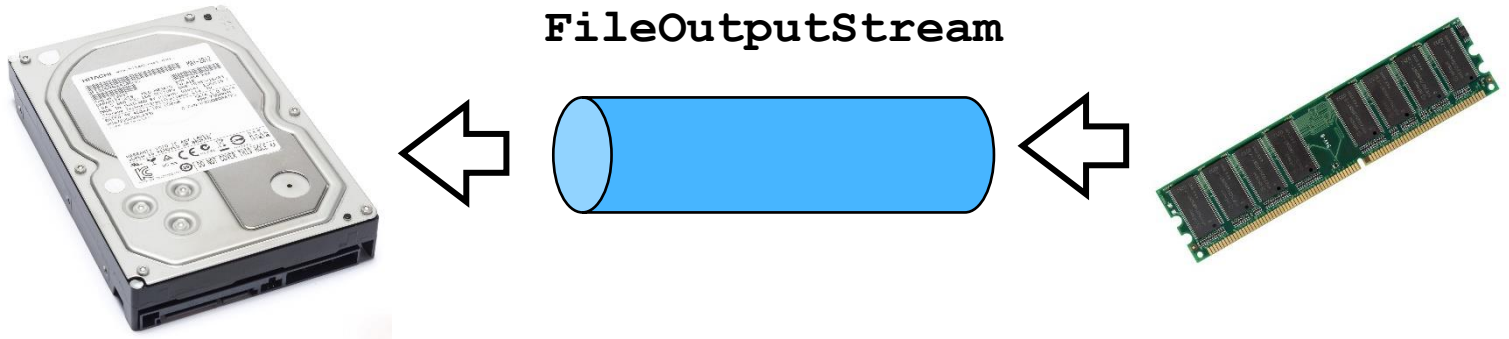
```
System.out.print("Informe o preço: ");

Scanner teclado = new Scanner(System.in);
double preco = Double.parseDouble(teclado.nextLine());

System.out.println("Valor digitado: " + preco);

File f = new File("c:\\dados.bin");
FileOutputStream fos = new FileOutputStream(f);
fos.write(preco);
```

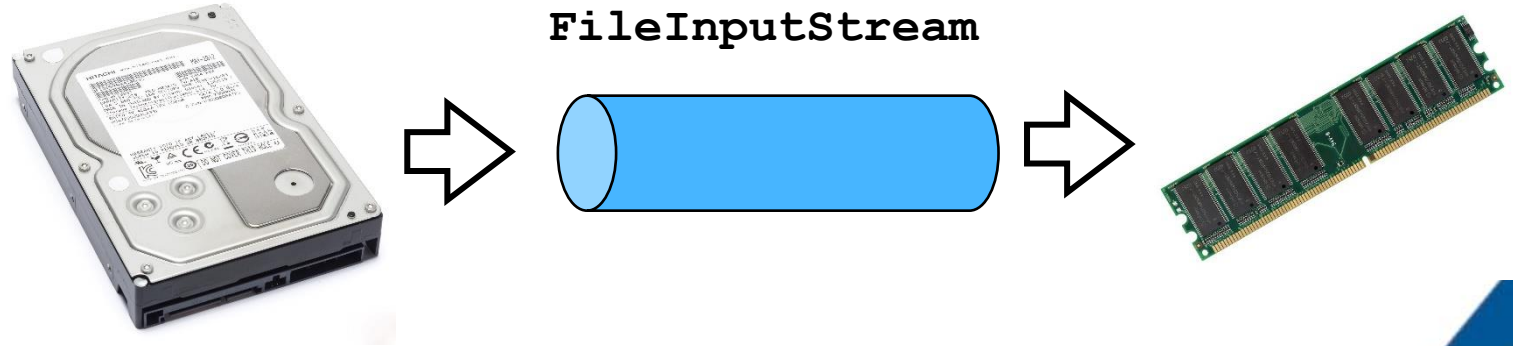
Esperado int
informado double

Como gravar o valor da variável **`preco`** no arquivo?

Stream

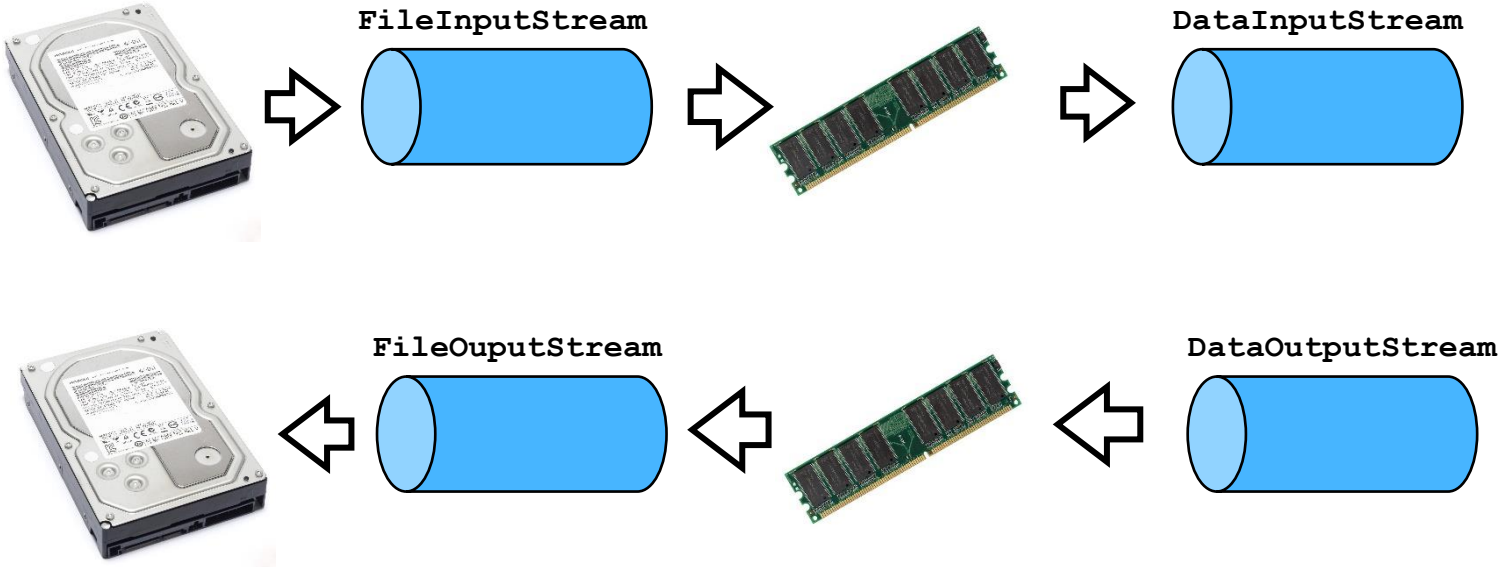
- Um **Stream** é um objeto que tanto obtém dados de uma fonte (como teclado, arquivo, rede) como grava dados (tela, arquivo, etc).



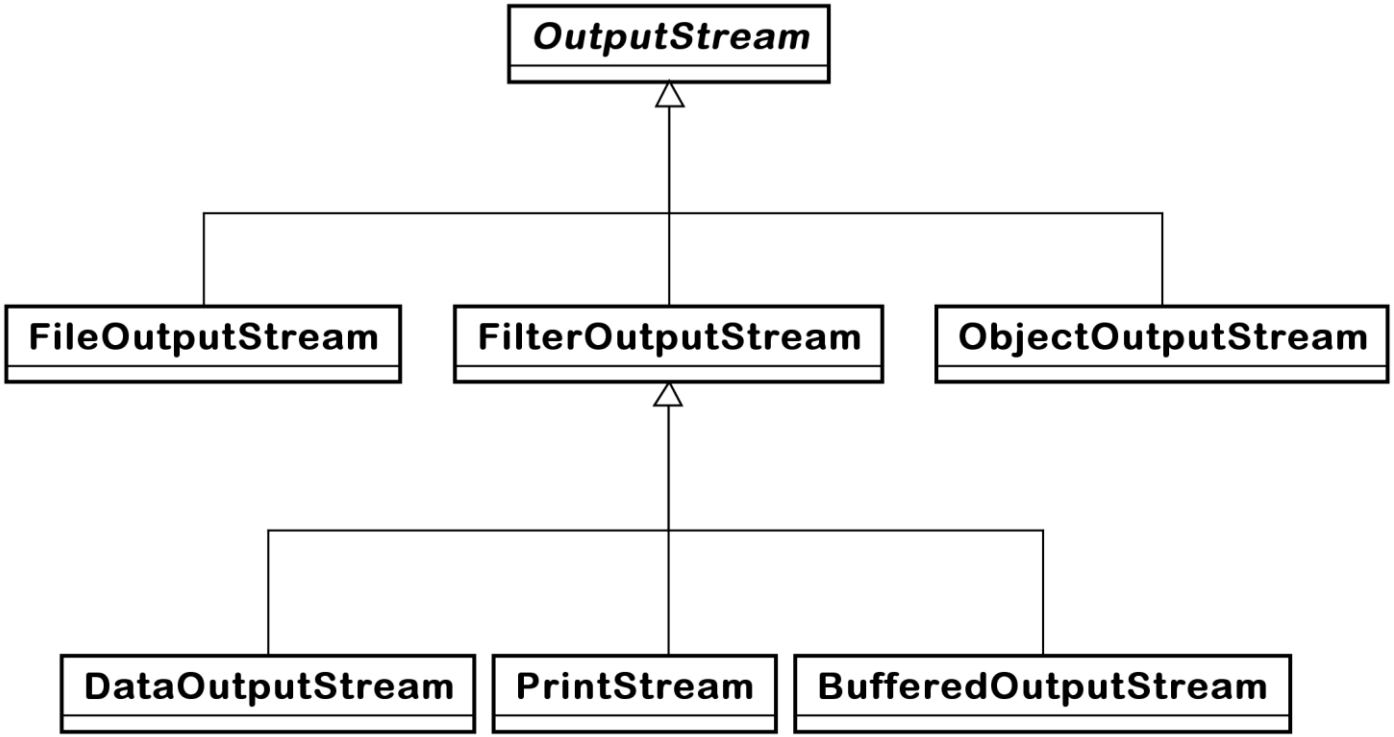


Stream

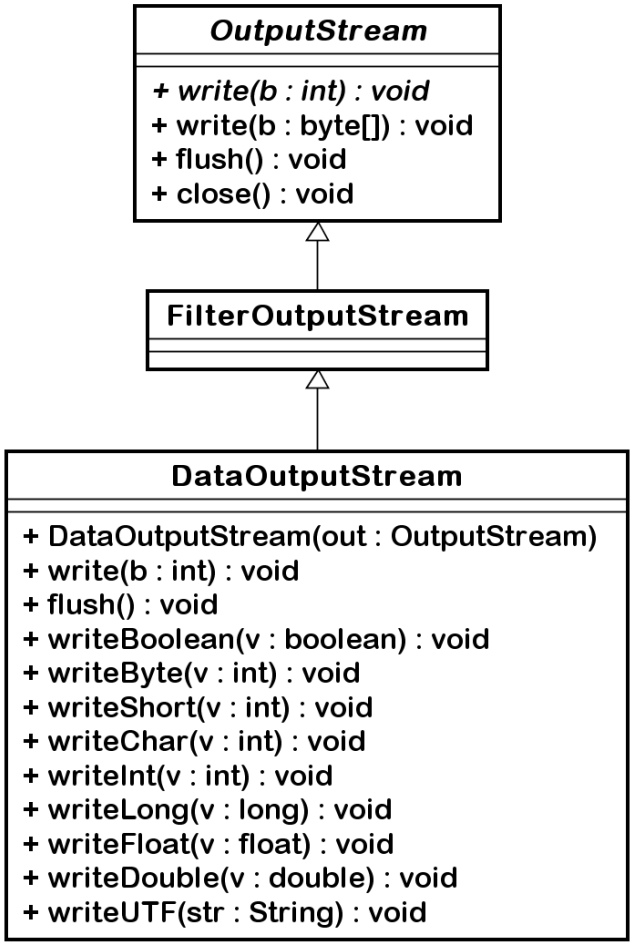
- É possível encadear outro objeto que estende *OutputStream* para criar outra forma alternativa de ler os dados



Hierarquia de classes para gravação de dados



DataOutputStream



- **`DataOutputStream`** oferece uma série de métodos para gravação de dados primitivos e string

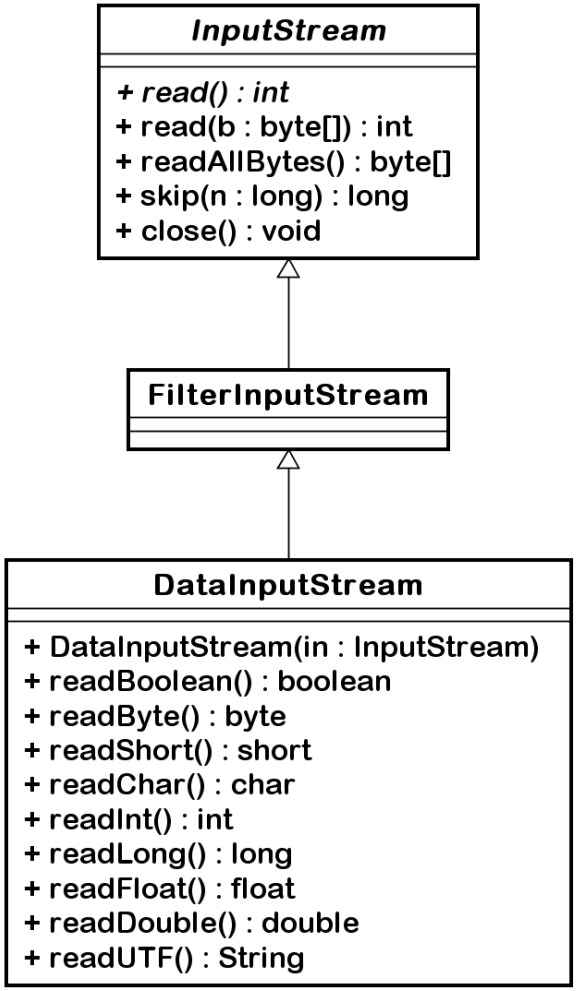
Exemplo

```
File f = new File("c:\\dados.bin");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos = new DataOutputStream(fos);
dos.writeDouble(preco);
dos.writeBoolean(true);
dos.writeUTF("FURB");
dos.close();
```

Observações

- Somente o objeto da classe **`DataOutputStream`** precisa ser fechado
  - Ao encadear streams, somente o objeto mais externo ao encadeamento precisa ser fechado

Leitura de arquivos



- **`DataInputStream`** oferece uma série de métodos para leitura de dados primitivos e string

- Após a leitura, o ponteiro é posicionado após a quantidade de bytes lidos.

Observações

- **`DataInputStream`** e **`DataOutputStream`** respectivamente, lê e grava tipos primitivos Java e Strings de forma independente de máquina

- Os dados devem ser lidos na mesma ordem em que foram gravados.

- A exceção `EOFException` é lançada quando há a tentativa de ler dados além do fim do arquivo

Exemplo

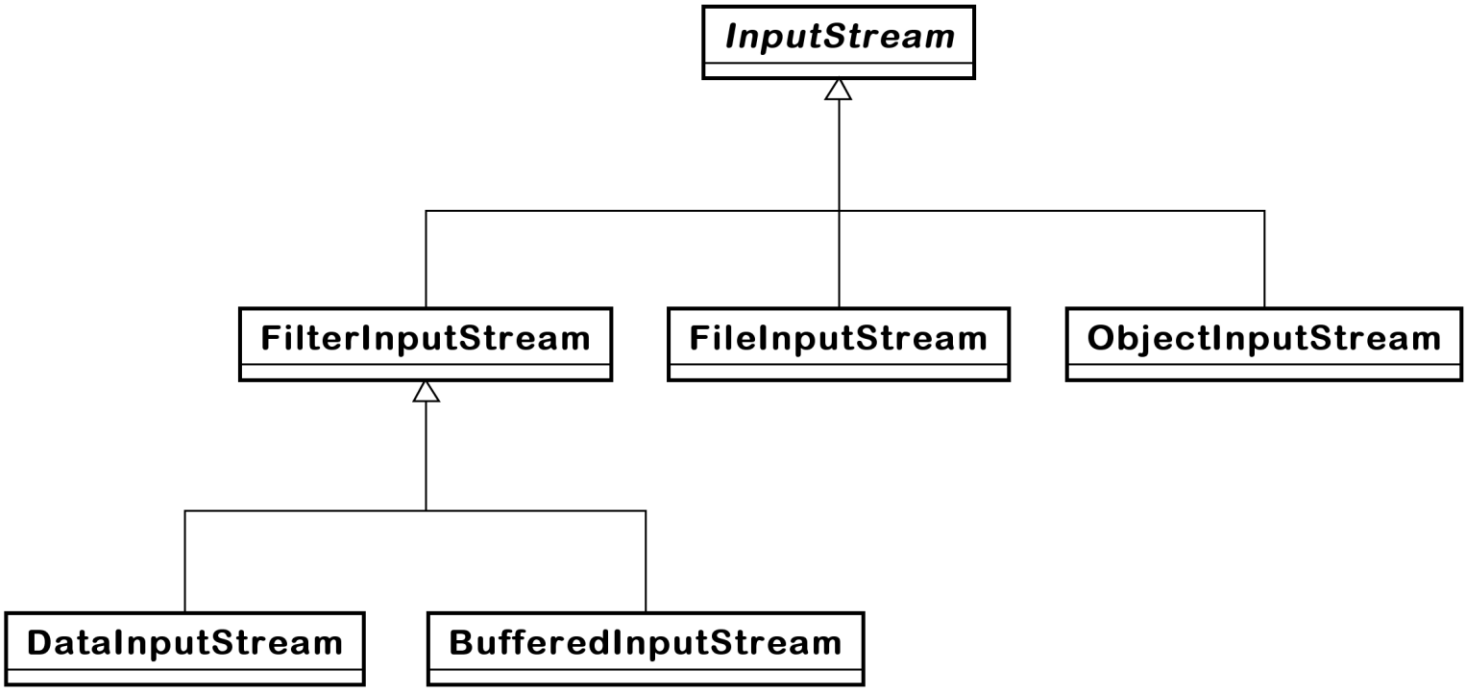
```
File f = new File("c:\\dados.bin");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos = new DataOutputStream(fos);
dos.writeDouble(preco);
dos.writeBoolean(true);
dos.writeUTF("FURB");
dos.close();
```

Exemplo

```
File f = new File("D:\\dados.bin");
FileInputStream fis = new FileInputStream(f);
DataInputStream dis = new DataInputStream(fis);
double preco = dis.readDouble();
boolean bool = dis.readBoolean();
String instituicao = dis.readUTF();
dis.close();
```

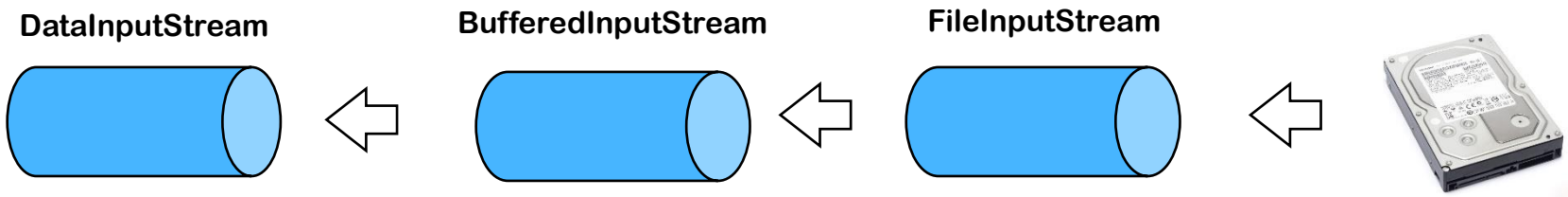
Buffer de dados na leitura/gravação

Leitura de arquivos binários

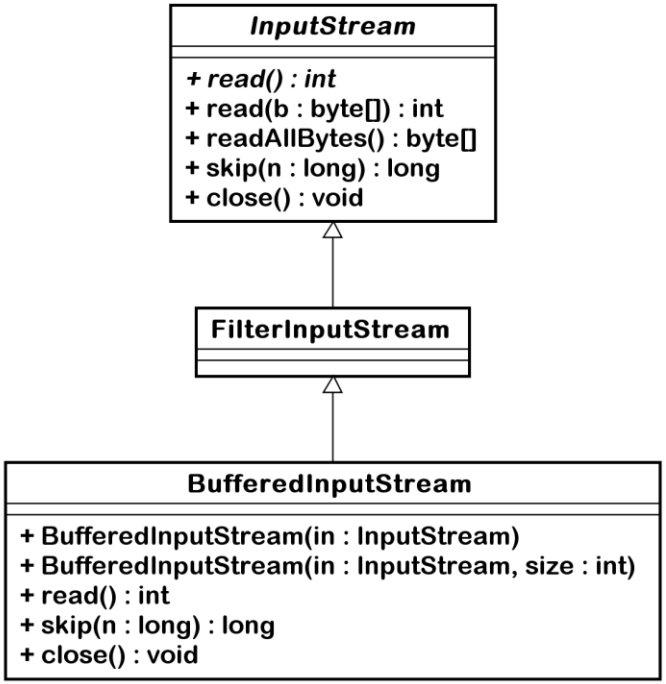


BufferedInputStream

- A classe **`BufferedInputStream`** é utilizada para reduzir a quantidade de acessos ao dispositivo de armazenamento, tornando a leitura mais otimizada

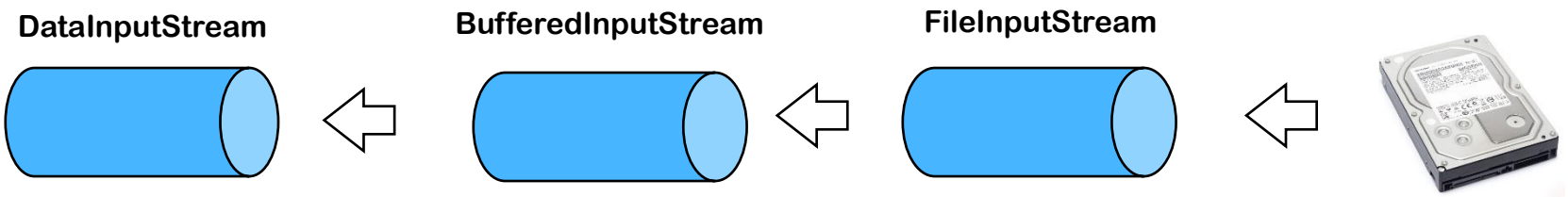


BufferedInputStream



Membro	Descrição
`BufferedInputStream(InputStream, size)`	Cria um objeto para ler dados de um *inputStream* Define o tamanho do buffer.
`BufferedInputStream(InputStream)`	Cria um objeto para ler dados de um *inputStream*. Utiliza um buffer de 8 KB.
`read()`	Lê um byte do *inputstream*. Retorna -1 quando atingir o final.
`skip()`	Salta para uma posição a frente do *inputstream*, em relação à posição atual. Deve ser um valor positivo
`close()`	Fecha o *inputstream*

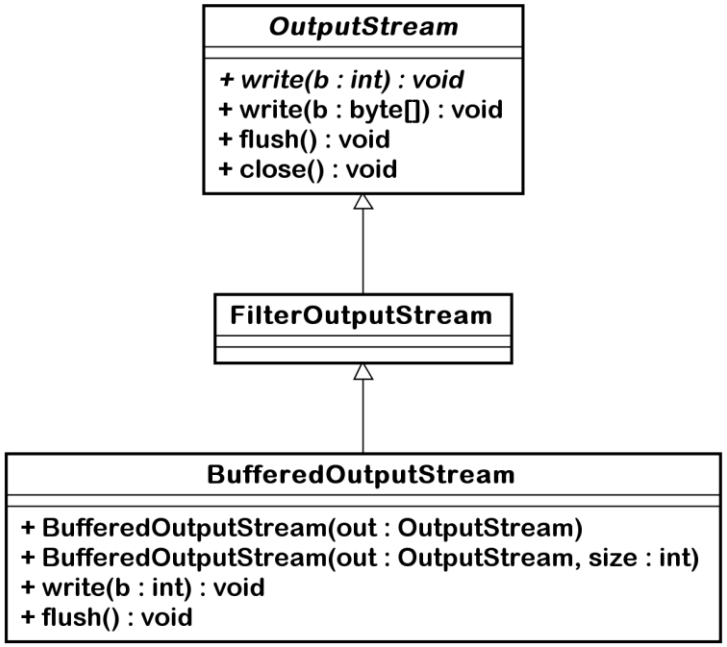
Exemplo:




```
File arquivo = new File("D:\\dados.dat");

FileInputStream fis = new FileInputStream(arquivo);
BufferedInputStream bis = new BufferedInputStream(fis);
DataInputStream dis = new DataInputStream(bis);

dis.close();
```

Neste exemplo, deve-se utilizar os métodos do **`DataInputStream`** para ler os dados do stream

BufferedOutputStream



Membro	Descrição
`BufferedOutputStream(OutputStream, size)`	Cria um objeto para gravar dados em um *outputStream* Define o tamanho do buffer.
`BufferedInputStream(InputStream)`	Cria um objeto para gravar dados em um *inputStream*. Utiliza um buffer de 8 KB.
`write(int)`	Grava um byte no *outputstream*.
`flush()`	Grava os dados imediatamente