

Programação Orientada a Objetos

Departamento de Computação
Universidade Federal de Sergipe

Arquivos

Prof. Kalil Araujo Bispo
kalil@dcomp.ufs.br



Conteúdo

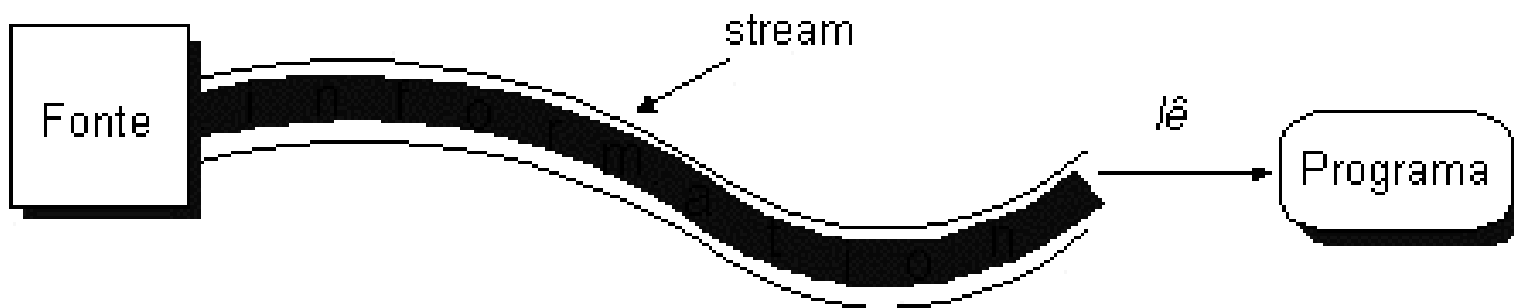
- Introdução
- Streams
- Streams de Caracteres
- Streams de Bytes
- Uso básico
- Escrevendo Caracteres
- Lendo Caracteres
- Escrevendo Bytes
- Lendo Bytes
- Serialização
- Console
- Acesso Randômico

Introdução

- Frequentemente os programas precisam trazer informações de uma fonte externa ou enviar informações para um destino externo
- Podem ser de vários tipos:
 - Caracteres
 - Imagens
 - Sons
 - Objetos

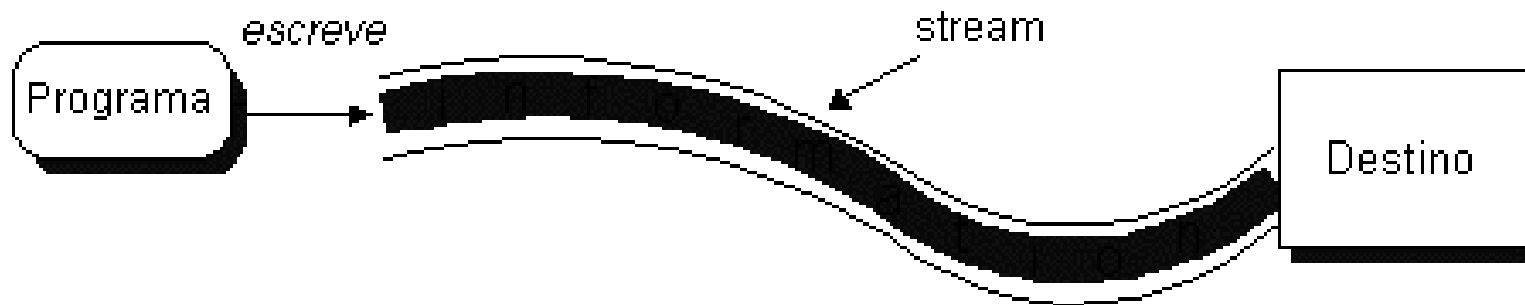
Streams

- A API de Entrada/Saída de Java é baseada em Streams
- Para trazer informações, um programa abre uma stream (corrente ou fluxo) de uma fonte de informações (um arquivo, memória ou socket) e lê a informação serialmente



Streams

- De forma similar, um programa envia informações para um destino externo abrindo uma stream para o destino e escrevendo as informações de forma serial



Streams

- Quando se trabalha com streams, não importa de onde a informação esta vindo ou para onde vai e também não importa que tipo de dado está sendo escrito ou lido
- A leitura e a escrita seguem sempre a forma

// Leitura

- 1) Abrir uma stream
- 2) Enquanto há informações
 - 2.1) Ler a informação
- 3) Fechar a stream

// Escrita

- 1) Abrir uma stream
- 2) Enquanto há informações
 - 2.1) Escrever a informação
- 3) Fechar a stream

Streams

- O pacote `java.io` contém uma coleção de classes para trabalhar com streams que suportam esses algoritmos de leitura e escrita
- Há 2 hierarquias de classes baseadas nos tipos de dados sobre os quais operam
 - Caracteres
 - Bytes

Streams de Caracteres

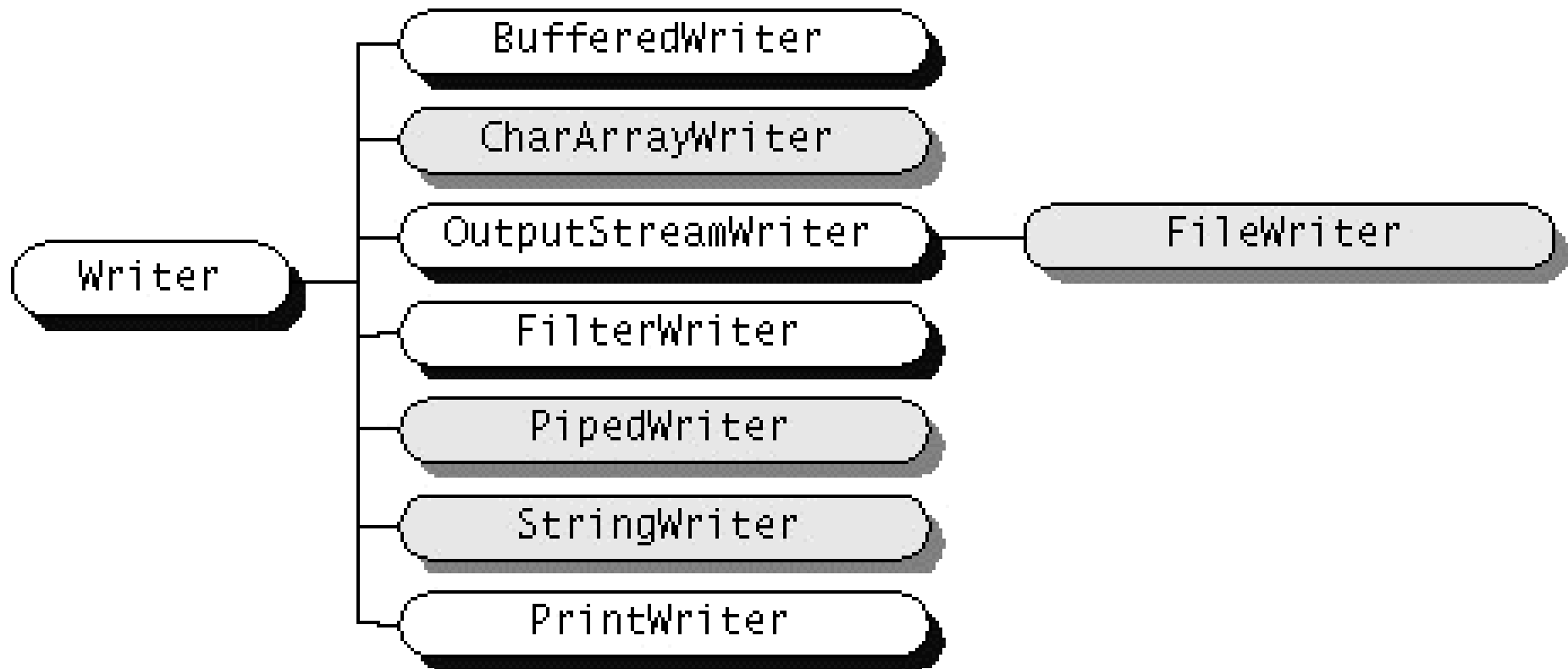
- As classes `java.io.Reader` e `java.io.Writer` são as superclasses abstratas para as streams de caracteres no pacote `java.io`
 - **Reader** define métodos e traz uma implementação parcial para leitores – Streams que leem caracteres
 - **Writer** define métodos e traz uma implementação parcial para escritores – Streams que escrevem caracteres

Streams de Caracteres

- As subclasses de Reader e Writer implementam streams especializadas e são divididas em duas categorias:
 - Leem de e escrevem para fontes de dados (cinza)
 - Executam algum tipo de processamento (branco)

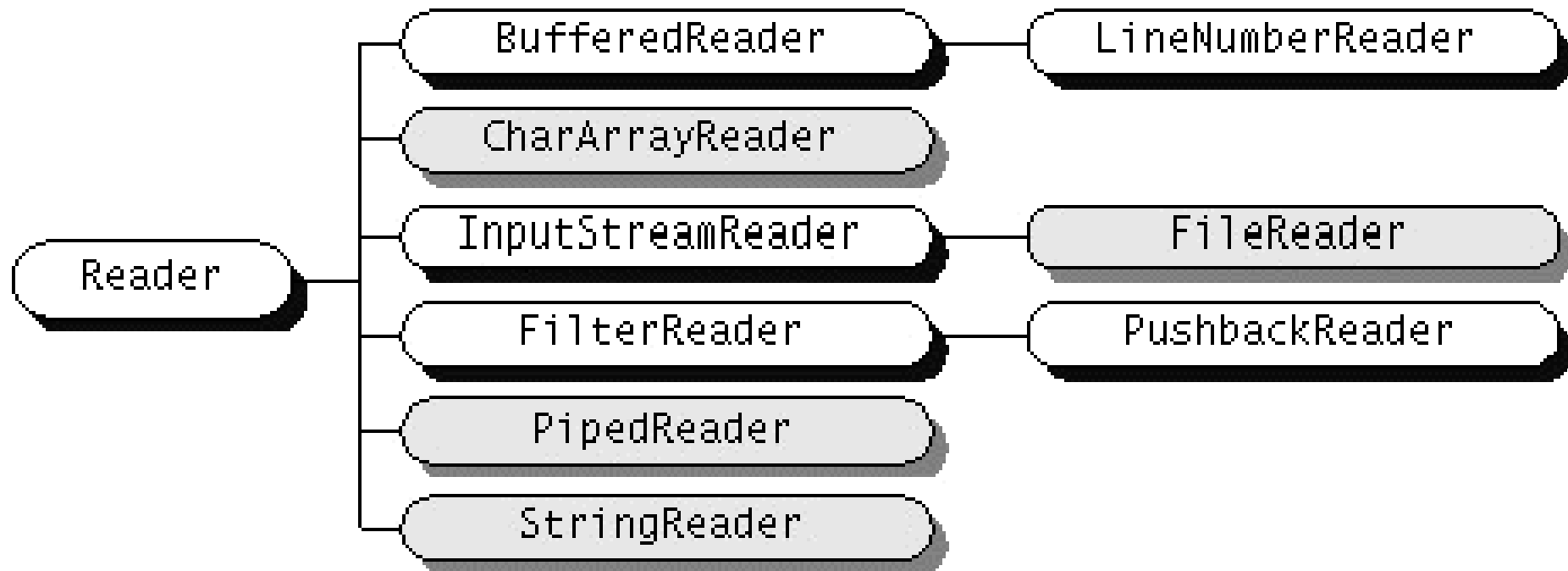
Streams de Caracteres

- Descendentes de Writer



Streams de Caracteres

- Descendentes de Reader



Streams de bytes

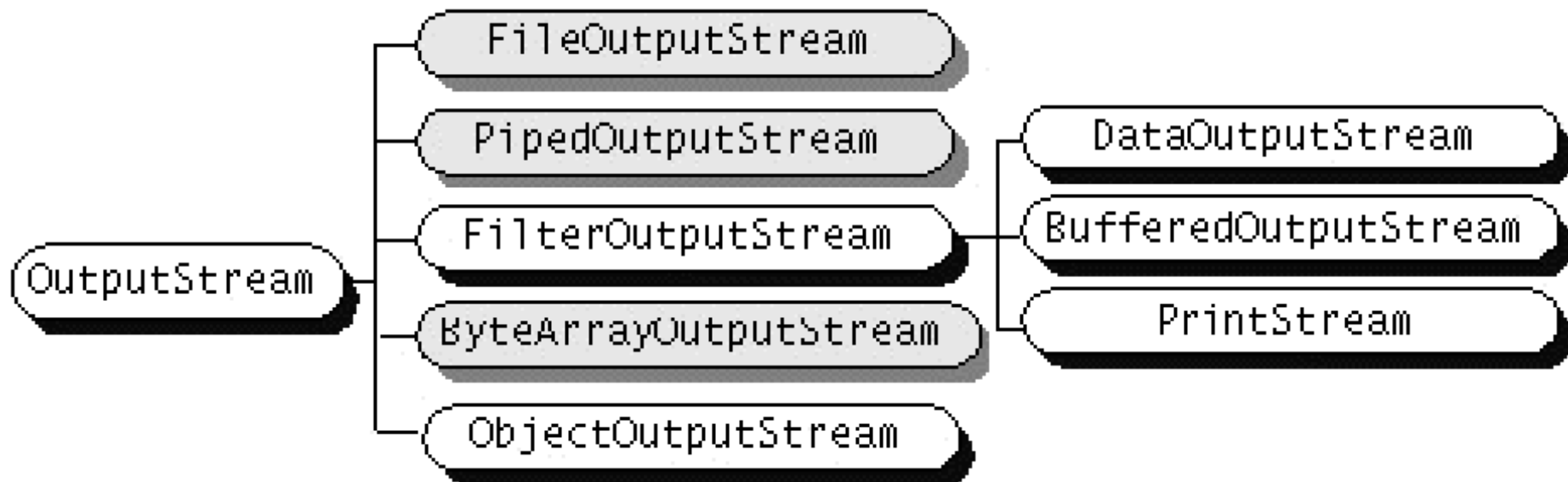
- As classes descendentes de `java.io.InputStream` e `java.io.OutputStream`, implementam a leitura e escrita em streams de bytes
- `InputStream` e `OutputStream` definem métodos e trazem alguma implementação para streams de entrada (streams que lêem bytes) e streams de saída (streams que escrevem bytes)
- Essas streams são usadas tipicamente para ler e escrever dados binários (imagens, sons, etc)

Streams de bytes

- Da mesma forma que Reader e Writer, as subclasses de InputStream e OutputStream dividem-se em duas categorias:
 - Leem de e escrevem para fontes de dados (cinza)
 - Executam algum tipo de processamento (branco)

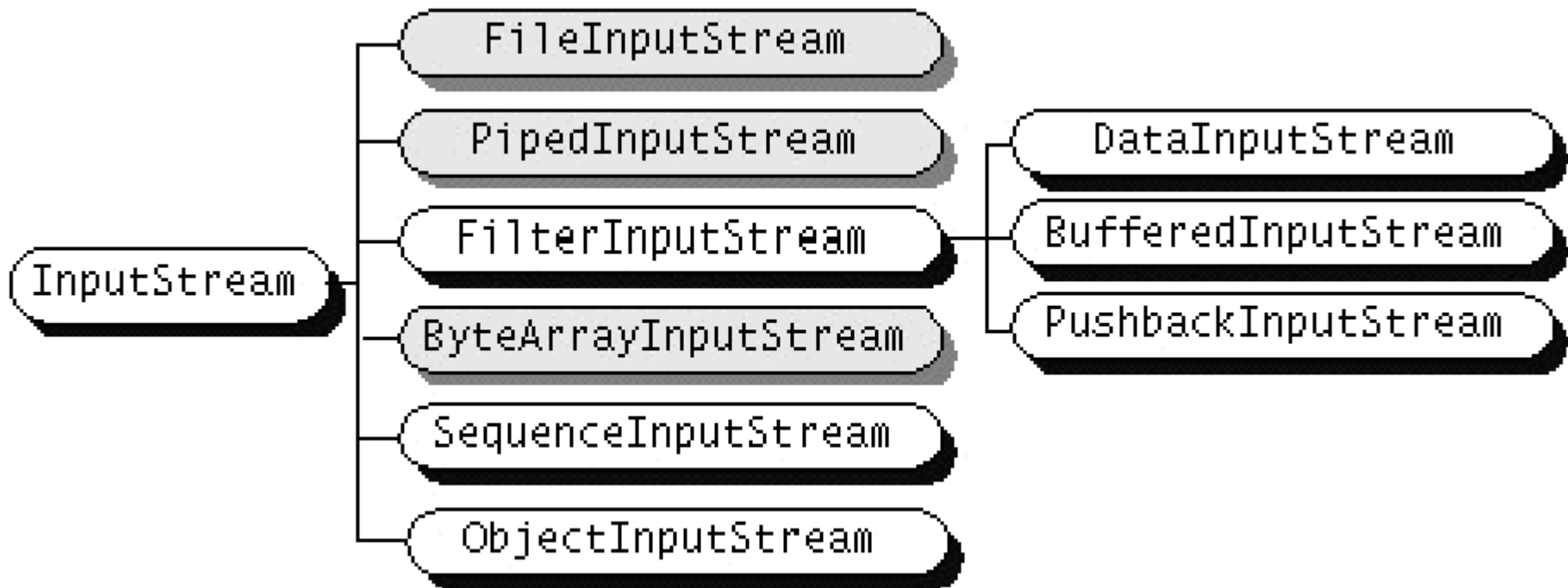
Streams de bytes

- Descendentes de OutputStream



Streams de bytes

- Descendentes de InputStream



Uso Básico

- Writer e OutputStream também definem métodos similares

- Principais métodos de Writer

```
void write(int c)
void write(char buf[])
void write(char buf[], int offset, int length)
void write(String str)
void write(String str, int off, int len)
```

- Principais métodos de OutputStream

```
void write(int c)
void write(byte buf[])
void write(byte buf[], int offset, int length)
```


Uso Básico

- Reader e InputStream definem métodos similares para tipos de dados diferentes

- Principais métodos de Reader

```
int read()  
int read(char buf[])  
int read(char buf[], int offset, int length)  
boolean ready()
```

- Principais métodos de InputStream

```
int read()  
int read(byte buf[])  
int read(byte buf[], int offset, int length)  
int available()
```

Uso Básico

- Todas as streams estão automaticamente abertas quando criadas
- Para fechar uma stream explicitamente chamando seu método `close()`
- O método `flush()` pode ser usado para forçar a descarga do buffer de saída (escrita)

Uso Básico

- **Reader e InputStream têm métodos para:**
 - Marcar um ponto em uma stream (mark)
 - Pular entradas (skip)
 - Reajustar a posição corrente (reset)
- **Só é possível em streams que suportam essa funcionalidade, o que é indicado por `markSupported`**

Uso Básico

- Interfaces

- **DataOutput** e **DataInput**

- Definem métodos `write<tipo>` e `read<tipo>` para tipos primitivos, array de Bytes e String
 - Define um método `readLine`

- **ObjectOutput** e **ObjectInput**

- Estendem, respectivamente, **DataOutput** e **DataInput** para permitir a escrita e leitura de objetos
 - Definem, respectivamente, os métodos `writeObject` e `readObject`

Uso Básico

- A classe `java.io.File`

- Representa um arquivo no sistema de arquivos
- Pode-se consultar informações sobre o arquivo (como o caminho completo, por exemplo)
- Para criar um objeto `File` o mais comum é utilizar o construtor que recebe uma `String` contendo o caminho (relativo ou absoluto) do arquivo

Uso Básico

- Várias classes do pacote `java.io` aceitam um objeto `File` como argumento no seu construtor para informar com qual arquivo irão trabalhar
- Traz métodos para criar, remover e renomear
- Exemplo
 - `FileInfo`

Escrevendo Caracteres

- Descrição das classes

- **FileWriter**

- Classe de conveniência para escrever caracteres em arquivos
 - Estende OutputStreamWriter
 - Não define novos métodos

- **StringWriter**

- Escreve os caracteres em um StringBuffer (getBuffer e toString)

Escrevendo Caracteres

– CharArrayWriter

- Escreve os caracteres em array de caracteres (size, toCharArray, toString e writeTo)

– PipedWriter

- Trabalha em conjunto com PipedReader formando um Pipe
- O que é escrito no PipedWriter é lido pelo PipedReader a ele associado

Escrevendo Caracteres

- **Classes decoradoras**

- **BufferedWriter**

- Utiliza um buffer para tornar as operações mais rápidas
 - Cria um buffer para o Writer passado no construtor

- **FilterWriter**

- Classe abstrata para criar Streams de caracteres com filtro
 - Pode ser estendida, possivelmente definindo campos e mudando a implementação padrão dos seus métodos

Escrevendo Caracteres

- **PrintWriter**

- Escreve representações formatadas de tipos primitivos, arrays, objetos em Streams de caracteres
- Implementa vários métodos print e println

- **OutputStreamWriter**

- É uma ponte entre Streams de caracteres e bytes
- Writer que envia a saída para um stream de bytes

- **Exemplo**

- **WriterDemo**

Lendo Caracteres

- Descrição das Classes

- FileReader

- Classe de conveniência para ler caracteres de arquivos
 - Estende InputStreamReader
 - Não define novos métodos

- StringReader

- Utiliza como fonte de dados uma String
 - Comandos read farão leitura de seus caracteres

Lendo Caracteres

– CharArrayReader

- Lê os caracteres do array de caracteres passado no construtor
- Semelhante à classe StringReader

– PipedReader

- Trabalha em conjunto com PipedWriter formando um Pipe
- O que é escrito no PipedWriter é lido pelo PipedReader a ele associado

Lendo Caracteres

- **Classes decoradoras**

- **BufferedReader**

- Utiliza um buffer para tornar as operações mais rápidas
 - “Decora” o Reader passado no construtor

- **FilterReader**

- Classe abstrata para criar Streams de caracteres com filtro
 - Pode ser estendida, possivelmente definindo campos e mudando a implementação padrão dos seus métodos

Lendo Caracteres

– InputStreamReader

- É uma ponte entre Streams de caracteres e bytes
- Reader que obtém a entrada de um stream de bytes

– PushbackReader

- Estende FilterReader
- Permite ler um caractere da Stream (retirando-o) e depois colocá-lo de volta (se desejado)
- O construtor define o tamanho do buffer

Lendo Caracteres

- **LineNumberReader**

- Estende `BufferedReader`
- Controla o número da linha, baseando-se nos caracteres ‘\n’ e ‘\r’ ou um após o outro
- `getLineNumber` obtém o número da linha atual (começa com 0)
- `setLineNumber` altera o número da linha atual

- **Exemplo**

- `ReaderDemo`

Escrevendo bytes

- Descrição das classes

- `FileOutputStream`

- Stream de saída para gravar bytes em arquivo

- `ByteArrayOutputStream`

- Escreve os bytes em array de bytes (`size`, `toByteArray`, `toString` e `writeTo`)

- `PipedOutputStream`

- Trabalha em conjunto com `PipedInputStream` formando um Pipe
 - O que é escrito no `PipedOutputStream` é lido pelo `PipedInputStream` a ele associado

Escrevendo bytes

- **Classes Decoradoras**

- **FilterOutputStream**

- Classe para criar Streams de bytes com filtro
 - Pode ser estendida, possivelmente definindo campos e mudando a implementação padrão dos seus métodos

- **DataOutputStream**

- Implementa a interface DataOutput
 - Classe que permite escrever tipos primitivos e Strings em uma Stream de bytes
 - Define vários métodos write<tipo> (ex: writeLong)

Escrevendo bytes

– BufferedOutputStream

- Utiliza um buffer para tornar as operações mais rápidas
- Cria um buffer para a OutputStream passada no construtor

– PrintStream

- Escreve representações formatadas de tipos primitivos, arrays, objetos em Streams de bytes
- Implementa vários métodos print e println

Escrevendo bytes

- ObjectOutputStream
 - Implementa a interface ObjectOutputStream
 - Define métodos write<tipo>
- Exemplo
 - OutputStreamDemo

Lendo bytes

- **Descrição das classes**

- **FileInputStream**

- Stream de entrada para ler bytes de arquivo

- **ByteArrayInputStream**

- Lê os bytes do array de bytes passado no construtor

- **PipedInputStream**

- Trabalha em conjunto com PipedOutputStream formando um Pipe
 - O que é escrito no PipedOutputStream é lido pelo PipedInputStream a ele associado

Lendo bytes

- **Classes decoradoras**

- **SequenceInputStream**

- Representa a concatenação lógica das InputStream's passadas no construtor
 - A leitura será feita de forma que as várias Streams pareçam compor uma única Stream (em sequência)

- **FilterInputStream**

- Classe para criar Streams de bytes com filtro
 - Pode ser estendida, possivelmente definindo campos e mudando a implementação padrão dos seus métodos

Lendo bytes

– DataInputStream

- Implementa a interface DataInput
- Classe que permite ler tipos primitivos e Strings de uma Stream de bytes
- Define vários métodos read<tipo> (ex: readLong)

– BufferedInputStream

- Utiliza um buffer para tornar as operações mais rápidas
- “Decora” o InputStream passado no construtor

Lendo bytes

- **PushbackInputStream**

- Estende `FilterInputStream`
- Permite ler um byte da Stream (retirando-o) e depois colocá-lo de volta (se desejado)
- O construtor define o tamanho do buffer

- **ObjectInputStream**

- Implementa a interface `ObjectInput`
- Define métodos `read<tipo>`

- **Exemplo**

- `InputStreamDemo`

Serialização

- Transformação de um objeto em um conjunto de bytes que guarda os valores dos seus atributos de instância e metadados sobre o objeto
- Um objeto serializado pode ser passado através de streams. Isso é muito importante pois permite por exemplo:
 - Persistência de objetos em arquivos
 - Transmissão de objetos entre processos e máquinas através de sockets

Serialização

- Um objeto pode ser serializado quando implementa a interface `java.io.Serializable`
 - Não traz nenhum método (é uma interface de marcação), mas indica que a classe que a implementa suporta serialização
 - Os atributos da classe também precisam ser serializáveis
- Exemplo
 - Serializacao

Console

- A classe `java.lang.System` traz três atributos do tipo stream que representam
 - Entrada padrão (in - `InputStream`)
 - Saída padrão (out - `PrintStream`)
 - Saída de erro padrão (err - `PrintStream`)
- Traz 3 métodos para redirecioná-las
 - `setIn`, `setOut` e `setErr`
- Exemplo
 - RedirecionarSaídaArquivo

Acesso Randômico

- A classe `java.io.RandomAccessFile` é usada tanto para ler como para escrever arquivos
 - Ao contrário das outras classes de E/S disponíveis em `java.io`
- Permite a utilização de arquivos de acesso randômico (onde o acesso pode ser não sequencial)

Acesso Randômico

- **Implementa as interfaces `DataOutput` e `DataInput`**
 - Define vários métodos `read` e `write` para cada tipo de dado (`int`, `double`, `char`, `String`, etc)
- **Permite acesso para leitura ou leitura/gravação**
 - O modo “`r`” ou “`rw`” é passado no construtor

Acesso Randômico

- Além dos métodos de E/S, que implicitamente movem o ponteiro do arquivo, há 3 métodos que manipulam explicitamente o ponteiro do arquivo

```
// Move o ponteiro do arquivo para frente o número  
de // bytes especificados  
int skipBytes(int n)  
  
// Posiciona o ponteiro do arquivo exatamente  
// antes do byte especificado  
void seek(long pos)  
  
// Retorna a posição corrente  
long getFilePointer()
```

Acesso Randômico

- **Exemplo**
 - RandomAccessFileEscrever
 - RandomAccessFileLer