

Java

I/O::Fundamentos

PROF. MS. PETER JANDL JR | PROF. MS. NATHAN C. SILVA | PROF. MS. TÉLVIO ORRU

LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

CIÊNCIA DA COMPUTAÇÃO

UNIP [JUNDIAÍ]

A solid green horizontal bar at the bottom of the slide.

Arquivos e Streams

ARQUIVOS. SISTEMAS DE ARQUIVOS. STREAMS.
ORGANIZAÇÃO DO JAVA IO.

Aplicações & Persistência

Aplicações necessitam preservar seus dados entre as sessões de uso, pois sem isso tornam-se limitadas e repetitivas.

A preservação dos dados o que requer funções de armazenamento e recuperação, ou seja, um esquema de **persistência de dados**, como aquele oferecido pelos sistemas operacionais por meio de seus sistemas de arquivos.

Arquivos

Arquivos são conjuntos de dados armazenados na *memória secundária* de um sistema.

A *memória secundária* é formada por dispositivos de armazenamento não volátil, tais como unidades de disco rígido, disco óptico ou fita magnética; dispositivos de memória em estado sólido.

Os dados são preservados nestes dispositivos mesmo quando não estão energizados.

Desta maneira, os arquivos preservam dados que estão, usualmente, correlacionados.

Arquivos e Sistemas de Arquivos

A manipulação de **arquivos** envolve operações como:

- **Criação**
- **Escrita** (armazenamento de dados)
- **Leitura** (recuperação de dados)
- **Remoção**
- **Consulta e alteração** de metadados (nome, comentários, datas, atributos e permissões).

Estas operações, assim como as permissões para uso e controle dos arquivos são providos pelo *Sistema Operacional* por meio de um **sistema de arquivos**, que oculta os aspectos técnicos envolvidos no uso dos dispositivos da memória secundária.

Arquivos e Tipos de Arquivos

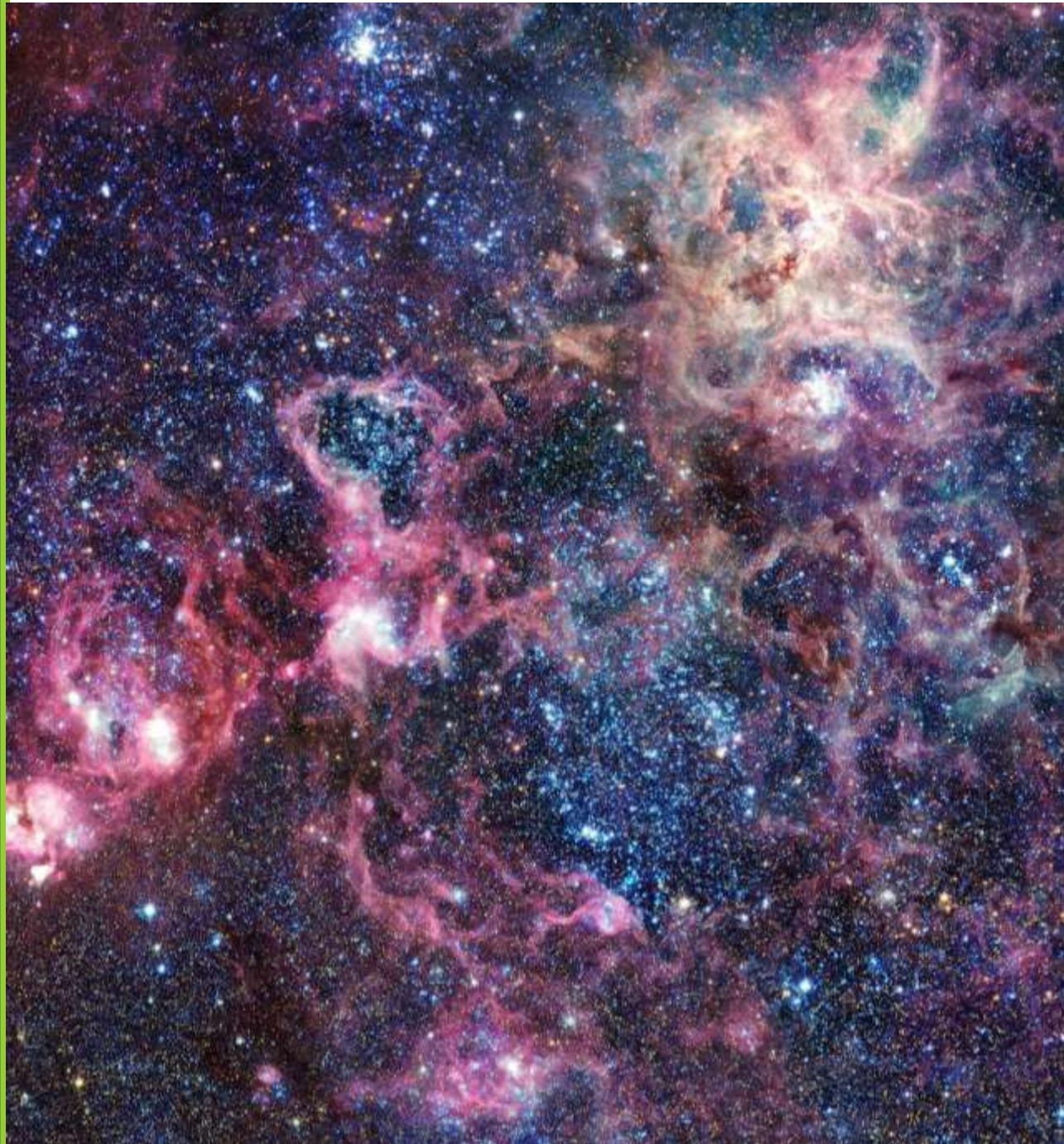
Arquivos de Texto

- conteúdo visualizável sem tratamento.

Arquivos Binários

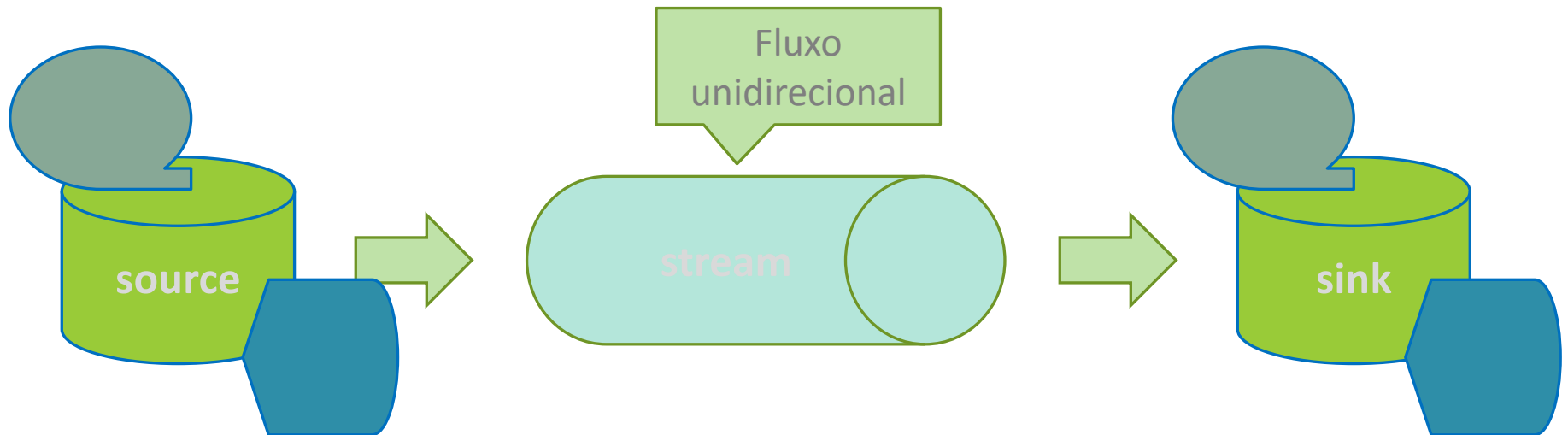
- Requer tratamento para visualização.

*Os
arquivos e
os sistemas
de
arquivos
são parte de
um cenário
maior.*



Streams

Um ***stream*** é um duto de dados, ou seja, um canal de comunicação que conecta um programa a uma fonte de dados (*data source*) para realizar operações de leitura, ou a um receptor de dados (*data sink*) para efetuar operações de escrita.



Arquivos, Streams, Dispositivos, Dados

PERSISTÊNCIA DE DADOS

Do ponto de vista de *armazenamento de dados*, os arquivos são um *mecanismo básico* de persistência oferecido por qualquer sistema operacional.

TRANSFERÊNCIA DE DADOS

Do ponto de vista de *transferência de dados*, os *streams* permitem a *conexão entre processos* (programas) e outros processos; entre processos e arquivos; ou mesmo entre processos e dispositivos.

Streams

Oferecem:

- operações **sequenciais** de manipulação de dados, ou seja, os dados devem ser manipulados a partir do seu início, até o ponto desejado, ou seu fim.
- O tratamento sequencial é semelhante ao de fitas de áudio, vídeo ou mesmo de dados (*backup*).

Permitem operar: memória, arquivos e dispositivos diversos (p.e. impressoras ou rede).

Alguns *streams* oferecem capacidades adicionais de processamento, tal como manipulação e transformação de dados.

Streams::uso geral

Rescrita requer processamento integral do arquivo.

LEITURA DE ARQUIVO

1. Stream é aberto (requer existência do arquivo)
2. Operação de leitura
3. Enquanto for desejado
E
Houverem dados
 - (nova) operação de leitura
4. Fechamento do arquivo

Arquivo não precisa ser lido integralmente!

ESCRITA DE ARQUIVO

1. Stream é aberto (se arquivo existe: reescrito – padrão – ou anexado; se não existe é criado)
2. Operação de escrita
3. Enquanto for desejado
 - (nova) operação de escrita
4. Fechamento do arquivo

Anexação não altera conteúdo já existente.

API Java I/O

API Java I/O

Framework de classes destinado a manipulação de **streams**.

Oferece dois grandes grupos de streams:

- Streams orientadas a **caractere**, destinadas a manipulação de arquivos (com conteúdo) de texto.
- Streams orientadas a **byte**, voltadas para o tratamento de arquivos de conteúdo binário.

A maior parte dos recursos desta API, que são classes, interfaces e exceções, se encontra no pacote **java.io**.

Java I/O::Streams para escrita/saída de dados

SAÍDA ORIENTADA A BYTES

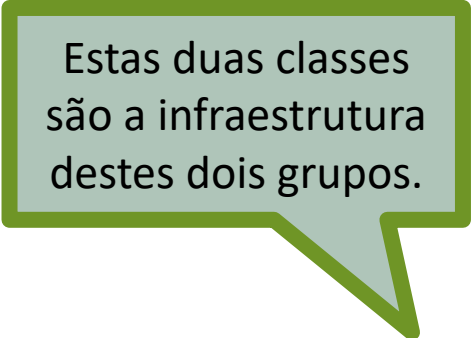
Classe **java.io.OutputStream**

Classe abstrata que representa um *stream* de saída composto de bytes.

SAÍDA ORIENTADA A CARACTERES

Classe **java.io.Writer**

Classe de infraestrutura para *streams* de escrita orientados a caractere.



Estas duas classes são a infraestrutura destes dois grupos.

Java I/O::Streams para leitura/entrada de dados

ENTRADA ORIENTADA A BYTES

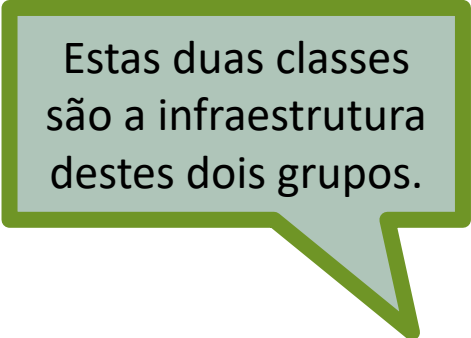
Classe **java.io.InputStream**

Classe abstrata que representa um *stream* de entrada composto de bytes.

ENTRADA ORIENTADA A CARACTERES

Classe **java.io.Reader**

Classe de infraestrutura para *streams* de leitura orientados a caractere.



Estas duas classes são a infraestrutura destes dois grupos.

Construção em Camadas

UMA ESTRATÉGIA PARA O USO EFETIVO DAS CLASSES DE JAVA.IO.

Construção em Camadas

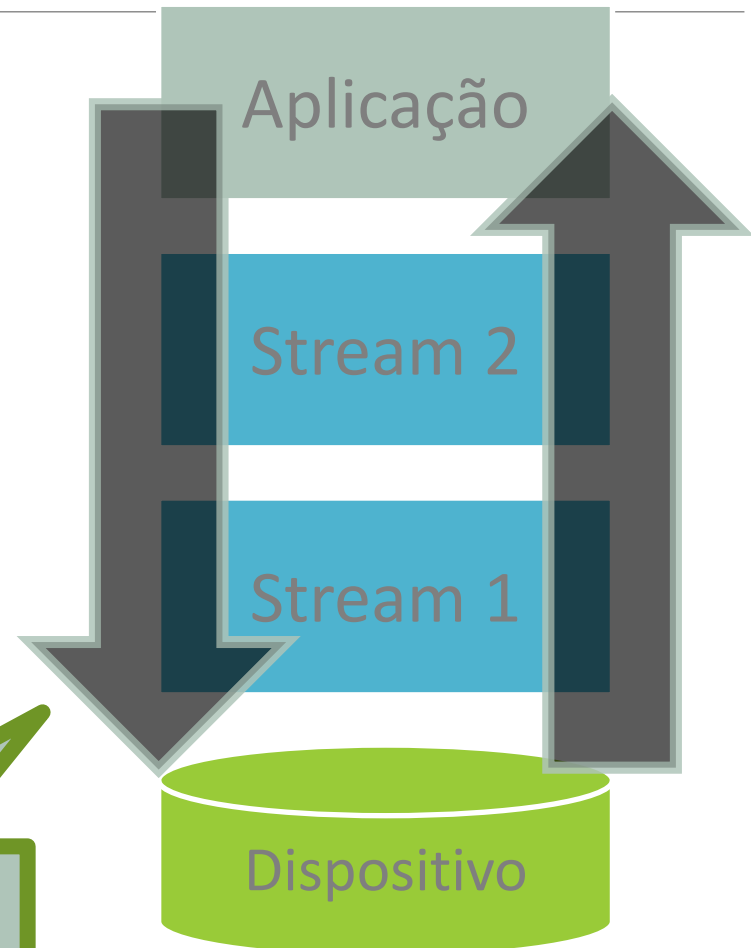
Estratégia necessária para uso apropriado dos **streams em Java**.

Primeiro stream (interno):
ligação com o dispositivo
(*source/sink*).

Segundo stream (externo): ligação
com a aplicação (*source/sink*).

O segundo stream toma o primeiro como um serviço de infraestrutura para oferecer uma conexão mais conveniente entre a aplicação e o dispositivo.

Cada par de camadas é usado **num único sentido**.



Tratamento Avançado de Erros::Exemplo

Muitas linguagens de programação modernas usam o conceito de exceção.

A **regra** é o código que deveria ser executados sem problemas.

Uma **exceção** é lançada quando ocorre um erro na execução da regra.

O **try/catch** é a construção do Java para tratamento das exceções.

```
try {
```

*Bloco onde se coloca
o código da **regra**
(processamento normal).*

```
} catch (IOException ioe) {
```

*Bloco onde se coloca
o código do tratamento da **exceção**
(processamento excepcional).*

```
}
```

Construção em Camadas::Exemplo

Pode existir uma classe adicional, como uma camada intermediária, convertendo, adaptando ou otimizando tal associação.

Exemplo ao lado é exemplo de leitura de dados na forma de uma construção em três camadas com tratamento mínimo de exceções.

```
try {  
    // constrói stream para leitura de arquivo  
    DataInputStream in = new DataInputStream (  
        new BufferedInputStream (  
            new FileInputStream( "nome.ext" ));  
    // enquanto existem dados  
    while (in.available() > 0) {  
        // lê dados com metodos específicos  
        System.out.println(in.readInt());  
    }  
    in.close(); // fecha o stream de leitura  
} catch (IOException exc) {  
    // mensagem de erro  
    System.out.println("Erro de IO");  
    // exibe detalhes sobre erro  
    exc.printStackTrace();  
}
```

Streams para Saída

Classe

`java.io.Writer::`subclasses úteis

BufferedWriter

Stream para escrita de caracteres em alto nível.

PrintWriter

Stream para escrita formatada de caracteres em alto nível.

FileWriter

Stream para escrita de caracteres em baixo nível em um arquivo.

OutputStreamWriter

Ponte entre streams orientados a byte e orientados a caractere, que escreve caracteres codificando-os em bytes.

Exemplos

E01_ESCRITATEXTOBASICO.JAVA

Aplicação que cria um arquivo de texto com conteúdo pré-determinado pelo programa.

E02_ESCRITATEXTO.JAVA

Aplicação que permite criar arquivos de texto contendo uma série de números aleatórios.

Pode ser modificada para permitir anexação ao invés da reescrita do arquivo.

Classe

java.io.OutputStream::subclasses úteis

Estas subclasse permitem dirigir a saída para um array (de bytes), para arquivos binários ou para serializar objetos.

ByteArrayOutputStream

dados são escritos em um array comum de bytes.

FileOutputStream

dados são escritos em um arquivo existente no sistema de arquivos.

ObjectOutputStream

permite armazenar objetos serializados.

Classe

java.io.OutputStream::subclasses úteis

FilterOutputStream

Classe base para mecanismos de filtragem para streams de saída orientados a byte.

- **DataOutputStream**

Permite saída de dados de tipos primitivos (char, int, float etc.).

- **BufferedOutputStream**

Permite otimizar as operações de escrita, realizando-as em blocos e usando um buffer interno.

- **PrintStream**

Permite a produção de dados formatados para exibição.

Exemplos

E03_ESCRITABYTES.JAVA

Utiliza a estrutura em camada dos streams Java para construir uma aplicação capaz de criar um arquivo binário contendo uma quantidade arbitrária de pares de valores int e double.

E04_SERIALIZA.JAVA

Cria um objeto (do tipo **Produto**) e o salva (serializa), de maneira transparente, em um arquivo binário apropriado.

Streams para Entrada

Exemplos

E05_LEITURABYTES.JAVA

Aplicação capaz de efetuar a leitura de uma quantidade não definida de pares de valores int e double.

Funciona de maneira complementar ao programa **E03_EscritaBytes**, ou seja, lê o arquivo que ele produz.

E06_DESSERIALIZA.JAVA

Recupera objeto (do tipo **Produto**) serializado, de maneira transparente, a partir de um arquivo binário apropriado.

Funciona de maneira complementar ao programa **E04_Serializa**, ou seja, lê o arquivo que ele produz.

Classe

java.io.InputStream::subclasses úteis

Estas classes permite obter/ler dados a partir de arrays (de bytes), de objetos StringBuffer, de arquivos binários ou recupera objetos serializados.

ByteArrayInputStream

permite usar array existente como fonte de dados.

StringBufferInputStream

permite usar StringBuffer como fonte de dados.

FileInputStream

permite usar um arquivo (binário ou ASCII) como entrada.

ObjectInputStream

permite recuperar objetos serializados.

Classe

java.io.InputStream::subclasses úteis

FilterInputStream

oferece suporte para outras classes especializadas, tais como `DataInputStream` e `BufferedInputStream`.

- **`DataInputStream`**

oferece suporte para leitura direta de tipos primitivos (`char`, `int`, `long`, `float`, `double` etc.).

- **`BufferedInputStream`**

usa um buffer de entrada para aumentar eficiência as operações de leitura.

Classe

java.io.Reader::subclasses úteis

InputStreamReader

ponte entre streams orientados a byte e streams orientados a caractere, que lê bytes decodificando-os em caracteres.

FileReader

stream para leitura de caracteres de um arquivo.

Exemplos

E07_LEITURATEXTO.JAVA

Aplicação capaz de abrir um arquivo de texto e de exibir seu conteúdo no console.

É como um complemento dos programas **E01_EscritaTextoBasico** ou **E02_EscritaTexto**, pois lê os arquivos que eles produzem.

E08_EXIBETEXTO.JAVA

Aplicação GUI capaz de exibir texto num JTextArea não editável.

Acesso Aleatório

Acesso Aleatório

Arquivos com processamento randômico são, em geral, constituídos de registros (conjunto predefinido de dados que ocupam um tamanho fixo em bytes correspondente à soma dos tamanhos de suas partes - campos).

No Java os registros *não são* construções especiais, mas *apenas* a justaposição de um grupo de elementos declarados como campos de uma classe.

Acesso Aleatório no Java

Classe **java.io.RandomAccessFile** oferece suporte para leitura e escrita em posições aleatórias de um arquivo de dados.

Seu uso é bastante peculiar, pois não se integra às famílias **InputStream**, **OutputStream**, **Reader** ou **Writer**, tão pouco é usada em camadas.

Exemplo

E09_ACESSORANDOM.JAVA

Aplicação GUI capaz de criar arquivos de dados para teste com formato conhecido e conteúdo aleatório; exibir arquivos de teste existentes num JTextArea.

Também efetua a pesquisa por registros específicos no arquivo em uso, apresentando tais registros na linha de status da aplicação.

Informação sobre Arquivos e Diretórios

Classe java.io.File

Permite obter informações e realizar operações sobre arquivos e diretórios.

É capaz de tratar aspectos dependentes do sistema de arquivos da plataforma em uso (como os caracteres separadores de diretórios e de lista de caminhos).

As operações disponíveis permitem distinguir diretórios e arquivos; determinar sua existência; obter seus atributos; criar, eliminar e renomear diretórios; bem como determinar o tamanho de arquivos.

Exemplo

E10_JDIR.JAVA

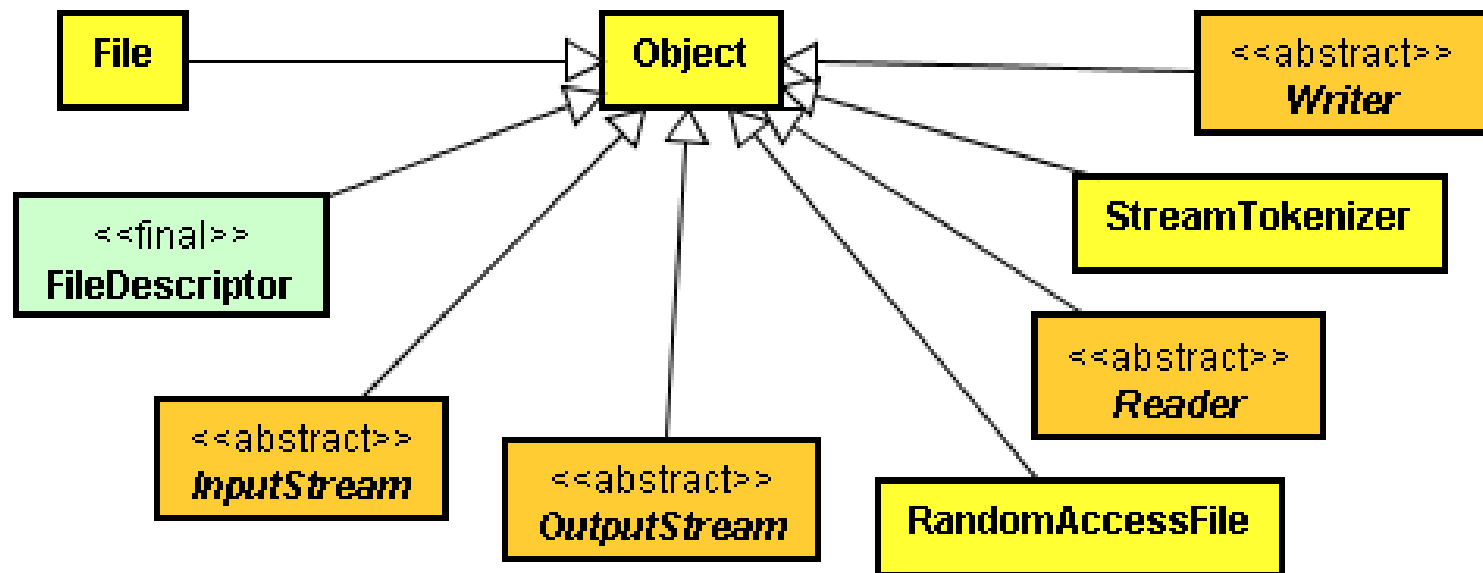
Aplicação de console que exibe informações sobre o arquivo ou diretório cujo nome é fornecido.

E11_ARVORE.JAVA

Aplicação GUI que exibe conteúdo de diretório como árvore.

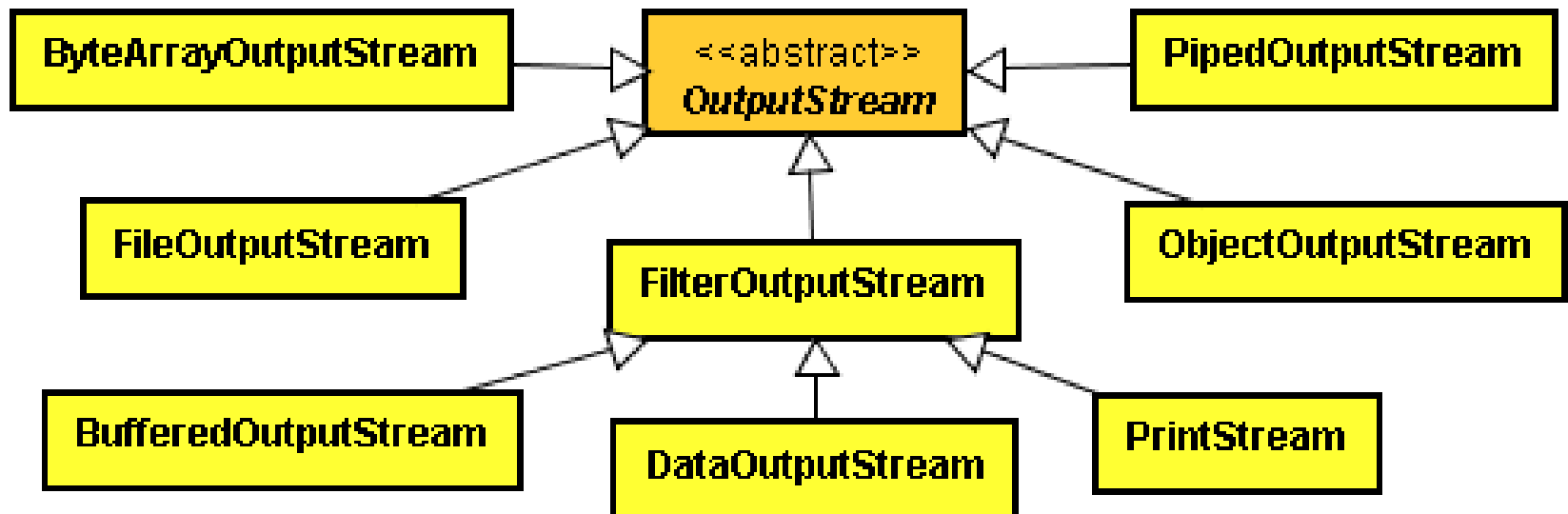
Tecnicidades

Hierarquia da API Java I/O



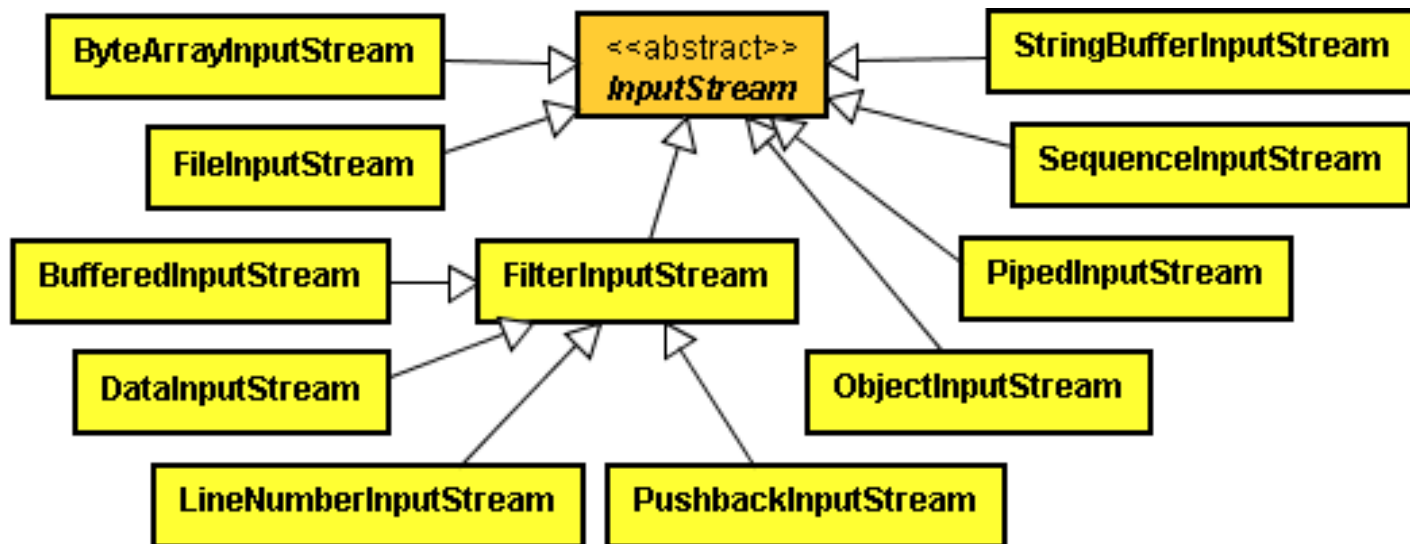
Classe

java.io.OutputStream::hierarquia

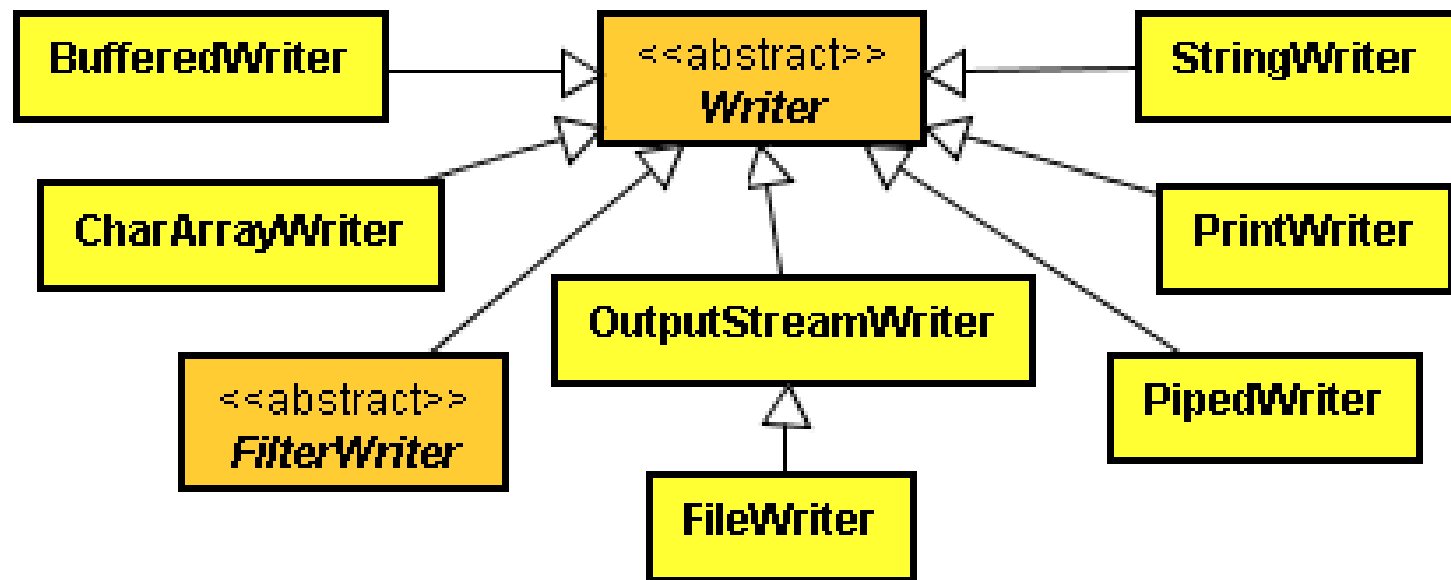


Classe

java.io.InputStream::hierarquia

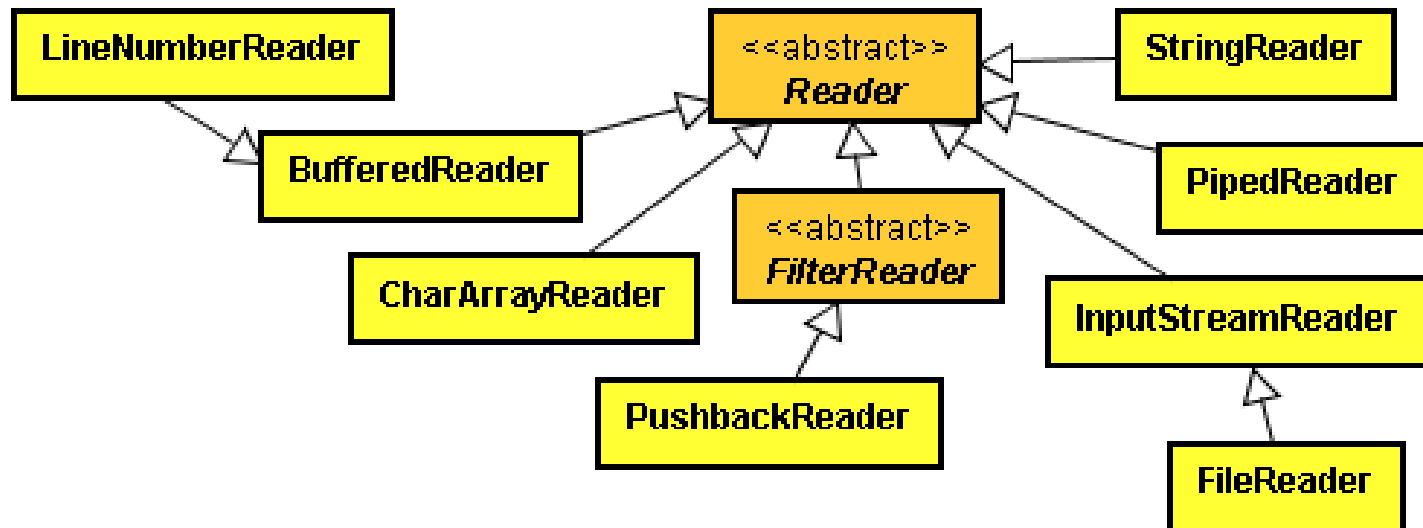


Classe java.io.Writer



Classe

java.io.Reader::hierarquia



ARM:Automatic Resource Management

Tratamento de erros adequado exige vários try/catch.

Construção trabalhosa e repetitiva.

A partir da versão 7, foi incorporado o ARM no try/catch transformando-o num *try-with-resources*.

```
try ( DataInputStream in = new DataInputStream (  
    new FileInputStream("nome.ext"))) {  
  
    // enquanto existem dados  
    while (in.available()>0) {  
        // lê dados c/ métodos específicos  
        System.out.println(in.readDouble());  
    }  
  
    // fechamento implícito do stream  
  
} catch (IOException exc) {  
    // mensagem de erro  
    System.out.println("Erro de IO");  
    // exibe detalhes sobre erro exc.printStackTrace();  
}
```

Exemplos

E12_ESCRITATEXTOARM.JAVA

Aplicação semelhante aos exemplo **E02_EscritaTexto**, que cria um arquivo de texto com conteúdo gerado aleatoriamente, mas com uso do **ARM**.

PICOEDITOR.JAVA

Aplicação GUI (muito simples) que permite criar arquivos de texto com o conteúdo de um componente JTextArea.

Exemplos

E13_LEITURABYTESARM.JAVA

Reescrita de
E05_LeituraBytes com **ARM**.

E14_LEITURATEXTOARM.JAVA

Reescrita de
E07_LeituraTexto com **ARM**.

Exemplos

JCOPY.JAVA

Aplicação de console capaz de copiar arquivos de qualquer natureza.

Sugestões de Estudo

Sugestões de Estudo

Material extra e exercícios
para reforço dos conceitos
vistos.

Exercícios

- ❑ Resolver a lista de exercícios III.

Material Extra

- ❑ JANDL JR, Peter.
Java – Guia do Programador,
3ª Ed. São Paulo: Novatec, 2015.