File Input/Output

Camila Cavalcante abril/21 - Digital Innovation One

Sumário:

Motivação:

- Praticamente em toda aplicação temos troca de informações.
- Isso quer dizer que temos um fluxo de dados (*stream*) que entra na nossa aplicação e um fluxo de saída que sai da nossa aplicação, sendo assim realizada a troca de informações, através de *streams.*
- O java utiliza esse conceito de *stream* para tornar essas operações de entrada e saída mais rápidas.
- O pacote responsável por essa parte é o pacote *java.io*, logo iremos dar foco nas classes que compõe esse pacote.
- Eventualmente também veremos outros pacotes, como por exemplo: java.lang e java.util

Manipulação de dados em um Computador

- Computador processa informações como 0 e 1 (bit).
- Bit é a menor unidade do computador e é manipulado por circuitos.
- Nós programadores não trabalhamos com bits e sim com números, letras e símbolos especiais.
- Como computador só trabalha com 0 e 1, cada caractere é representado por um padrão de 0 e 1 (1-Byte: 8 bits).
- ASCII (American Standard Code for Information Interchange): é um código numérico que estabelece uma correspondência entre a linguagem binária que é entendida pelo computador e os símbolos que utilizamos para comunicar e é representados por 1 byte (8 bits).
- UNICODE: É um padrão que permite aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita existente. Possui mais de 130 mil símbolos mapeados representados por 2 bytes (16 bits).
- Arquivo na visão do sistema operacional é um conjunto de bytes que serve para descarregar dados.
- A organização dos bytes em um arquivo é uma visão criada pelo aplicativo. Por exemplo: arquivo ppt, arquivo ASCII, conjunto de registros em um arquivo de banco de dados e etc...
- Java vê arquivo como um fluxo sequencial de bytes.
- Cada arquivo acaba com um marcador de fim. O Java avisa ao programador através de exceção ou do valor devolvido pelo método que processa o arquivo.

Materiais Recomendadíssimos: **Unicode**, **Arquivos** e **Exceção java.io**

Streams

Vamos começar falando de *streams*:

- Em Java, *streams* são a sequência de dados lidos (*input*) de uma fonte e gravados (*output*) no destino.
- Um *stream* é uma entidade lógica que produz (*output*) ou consome (*input*) informações.
- Um *stream* é vinculado a um dispositivo físico pelo sistema de I/O Java.
- Esse fluxo de dados podem sair diretamente do nosso teclado, de um arquivo de texto, de um socket (falando de rede). Logo, este o fluxo pode ser descarregado (saída do fluxo de dados) no nosso console, ser escrito em um arquivo de texto que será salvo no nosso HD, ir para um banco de dados, exibido no console e etc.

Na próximo tópico deste artigo, vamos partir para a hierarquia do pacote *java.io*, mas antes, quero deixar claro um conceito importante relacionado a *input* e *output*:

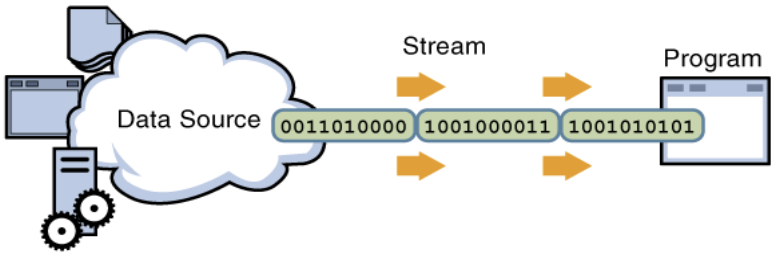


Figura 1: **Input**: Lendo informações em um programa - Fonte

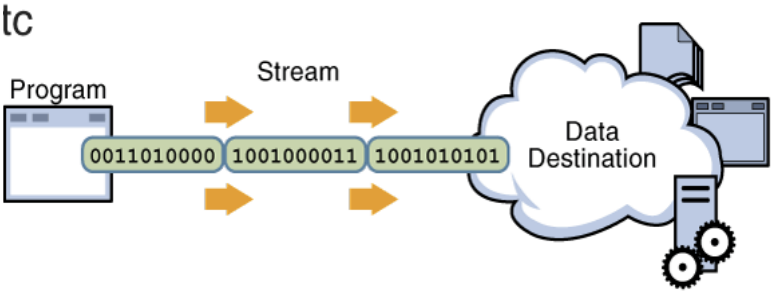


Figura 2: **Output**: escrevendo informações de um programa - Fonte

- Em relação de onde vem (input) o stream e para onde ele vai (output), pouco importa.
- No Java existem classes abstratas (mais genéricas) que nos ajudam na manipulação desses streams junto com classes concretas (mais específicas).
- Estas classes concretas são quem indicam de onde está vindo e para onde vai esse stream.
- Já as classes abstratas são quem recebem e enviam o stream.

- Quando queremos usar um stream, perguntamos primeiro:
    1. Em que direção o stream deve funcionar?
    2. Que tipo de informação será tratada?
    3. Qual é a fonte (na leitura) ou destino (na gravação) da informação?
    4. Qual processamento(s) intermediário(s) deve(m) ser feito(s) na informação?

Exemplo: Ler caracteres de um arquivo com bufferização

1. Direção: Ler
2. Tipo: Caracteres
3. Fonte: Arquivo
4. Processamento: Bufferização

Independente da origem ou destino dos dados, o processo de leitura (input) e escrita (output) de dados são os mesmos e envolvem os seguintes passos:

Leitura (read)

1. Abre um Stream
2. while(existe_informação)
   2.1 lê_informação
3. Fecha a Stream

Escrita(write)

1. Abre um Stream
2. while(existe_informação)
   2.1 escreve_informação
3. Fecha a Stream

Hierarquia java.io

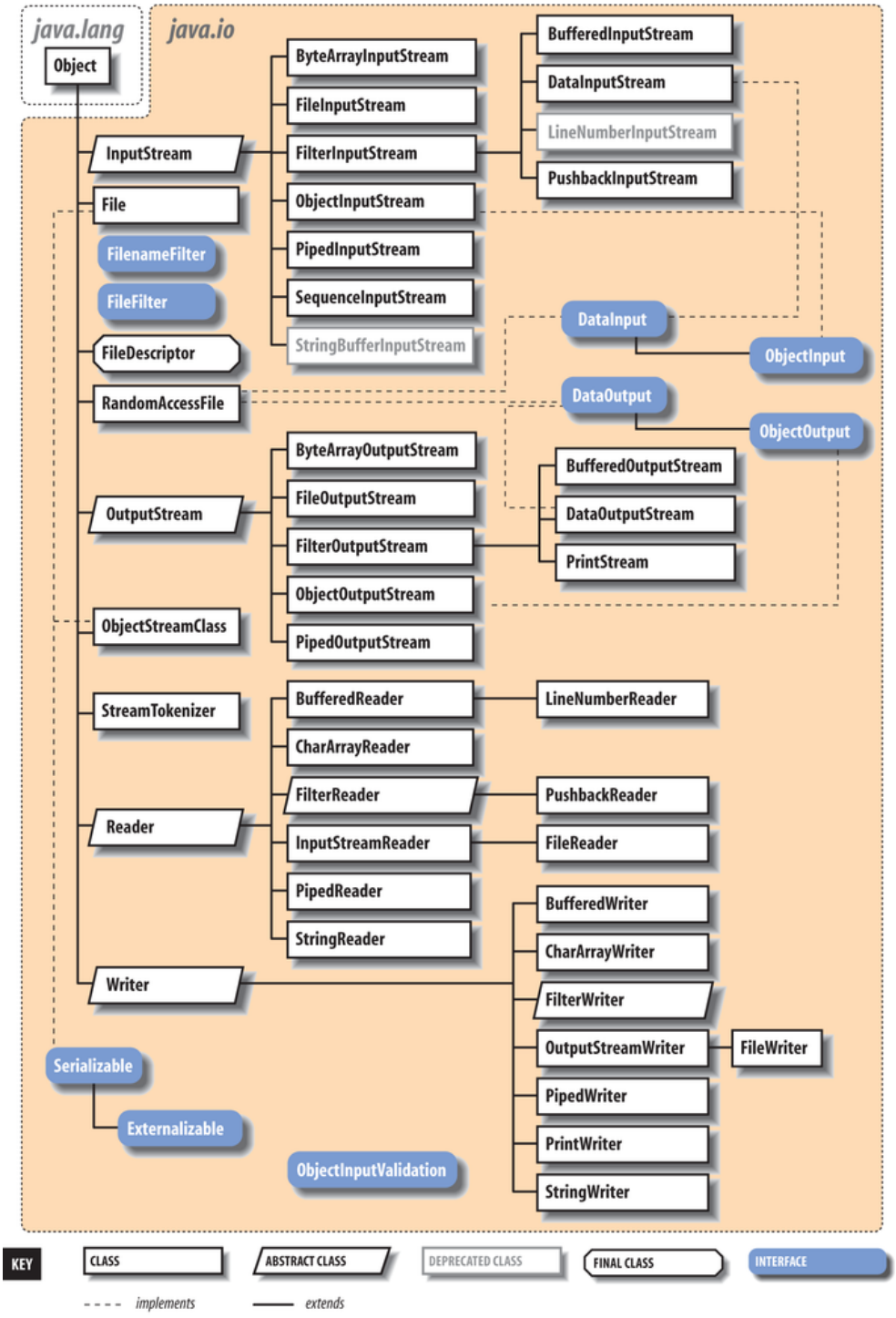
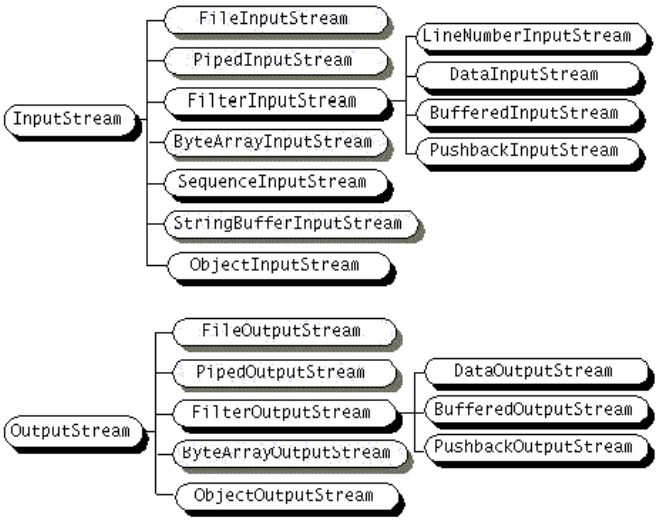


Figura 3: hierarquia java.io - Fonte

Classes que veremos no decorrer do curso:

- File: representa um ponteiro para um caminho. Isso não significa que o arquivo exista ou seja criado.

Byte Stream: dados na forma de bytes individuais (8-bits)



Classes abstratas

- InputStream: lida com fluxos de bytes de entrada: método read() lê um byte (8 bits) por vez.
- OutputStream: lida com fluxos de bytes de saída: método write() lê um byte (8 bits) por vez.

Classes concretas:

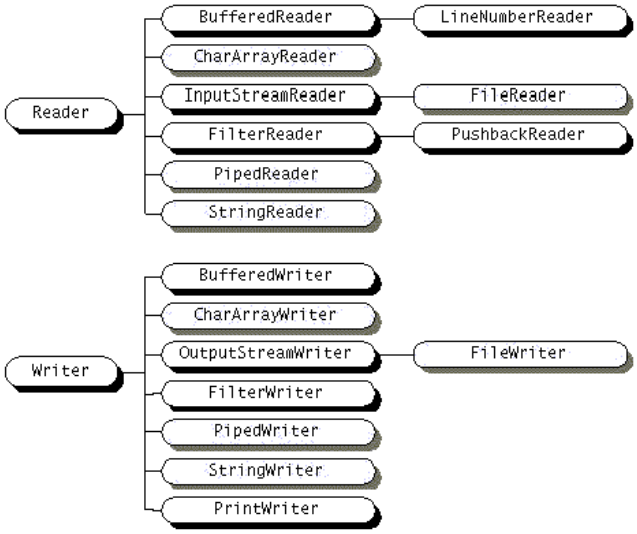
Input:

- FileInputStream: usadas para ler de um arquivo em disco.
- BufferedInputStream: adiciona funcionalidade a outro fluxo de entrada.
- ObjectInputStream: desserializa dados primitivos e objetos previamente escritos usando um ObjectOutputStream.
- DataInputStream: permite ler tipos primitivos de dados através de streams.

Output:

- FileOutputStream: usadas para escrever um arquivo em disco.
- BufferedOutputStream: adiciona funcionalidade a outro fluxo de saída.
- ObjectOutputStream: grava tipos de dados primitivos e gráficos de objetos Java em um OutputStream.
- DataOutputStream: permite escrever tipos primitivos de dados através de streams.
- PrintStream: grava em formato textual vários tipos de dados do java: boolean, char, int, etc.

Character Stream: dados na forma de character com 2 bytes (16-bits)



Classes abstratas:

- Reader: lida com fluxos de caracteres de entrada: método read() lê um caractere (16 bits) por vez
- Writer: lida com fluxos de caracteres de saída: método write() grava um caractere (16 bits) por vez

Classes concretas:

Input:

- InputStreamReader: é um filtro que converte bytes em chars.
- BufferedReader: permite a leitura de caracteres, linhas e texto de maneira mais eficiente
- FileReader: lida com leitura de caracteres de um arquivo.

Output

- OutputStreamWriter: é um filtro que converte chars em bytes.
- BufferedWriter: permite a escrita de caracteres, linhas e texto de maneira mais eficiente
- FileWriter: lida com escrita de caracteres de um arquivo.

Bibliografia:

1. https://www.programiz.com/java-programming/io-streams
2. https://www.devmedia.com.br/classe-inputstream-e-outputstream-em-java/32007
3. https://www.cs.auckland.ac.nz/references/java/java1.5/tutorial/essential/io/bytestreams.html
4. https://slideplayer.com/slide/9392500/#.YGc2_c_t-j0.gmail
5. https://www.cs.auckland.ac.nz/references/java/java1.5/tutorial/essential/io/bytestreams.html
6. https://www.devmedia.com.br/lendo-dados-de-txt-com-java/23221
7. http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/archivos/flujos.htm
8. http://docplayer.com.br/7734641-Java-2-standard-edition-entrada-helder-da-rocha-www-argonavis-com-br.html
9. https://www.cs.auckland.ac.nz/references/java/java1.5/tutorial/essential/io/datastreams.html
10. https://www.java64bit.com/java-filewriter-class.html
11. http://www.universidadejava.com.br/java/java-leitura-arquivo/
12. http://www.facom.ufu.br/~flavio/poo/files/2004-01/POO-JAVA%20IO.pdf
13. https://cs2113f18.github.io/java/6/
14. https://pt.slideshare.net/lineking/io-package/3
15. http://www.facom.ufu.br/~bacala/POO/Stream.pdf
16. https://pt.wikipedia.org/wiki/Unicode
17. http://www.dsc.ufcg.edu.br/~jacques/cursos/p2/html/ed/arquivos.htm