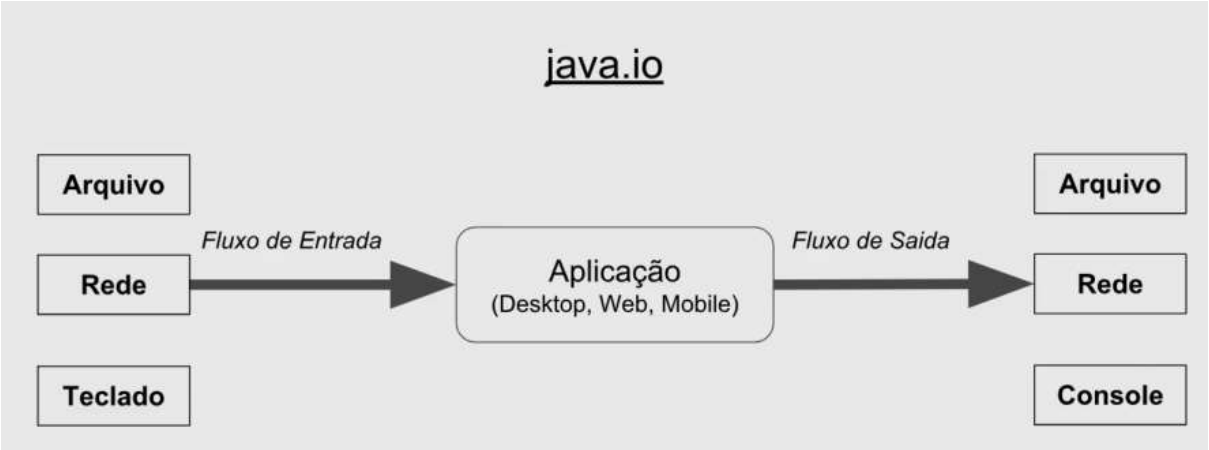
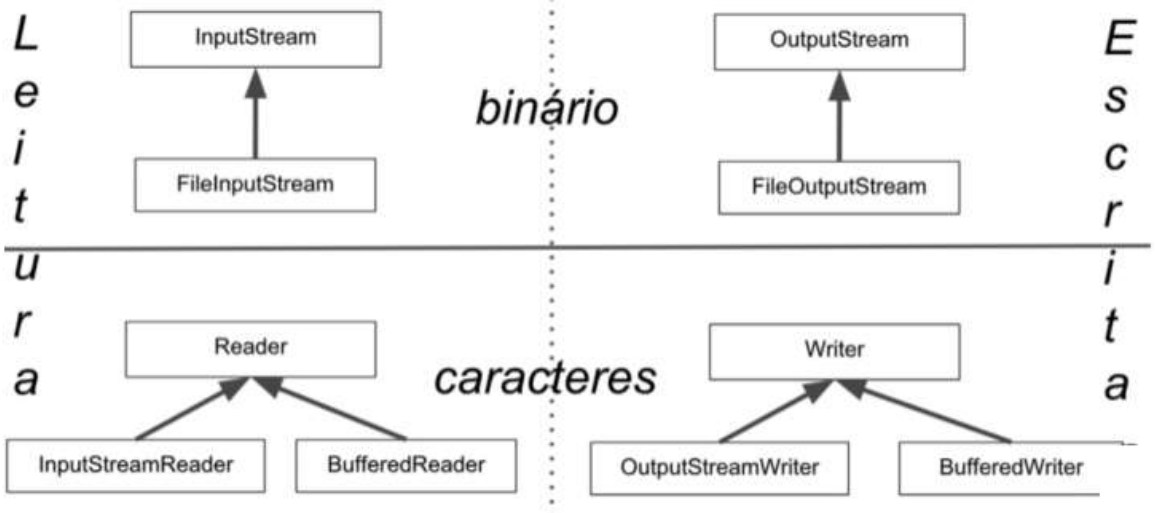


- STREAM e READER: Classes bases do Java.io e existem tanto para entrada quanto saída!!!
  - Stream: faz uma leitura de bits e bytes. P.S: Se precisamos ler uma imagem ou um PDF, por exemplo, utilizamos sempre o Stream.
  - Reader: também faz uma leitura, só que esta é focada nos caracteres. Se trabalharmos com um arquivo de texto, devemos utilizar o Reader.



- Leitura de arquivo:

➢ InputStream: é uma classe abstrata que representa o fluxo de dados binários.

InputStream fis = new FileInputStream("lorem.txt");

- Criamos o fluxo concreto com o arquivo, mas ainda binário. Lembrando que, *FileInputStream* (classe concreta) estende *InputStream* por isso podemos usar uma referência mais genérica.

➢ Reader: é uma classe abstrata que possui dois filhos concretos: a InputStreamReader e BufferedReader. O que ambas têm em comum é que são Readers, ou seja, compete à elas a leitura de caracteres.

Reader isr = new InputStreamReader(fis);

- conseguimos transformá-los em caracteres, mas apenas a contabilização. Lembrando que, *InputStreamReader* estende *Reader,* por isso podemos usar uma referência mais genérica.

➢ BufferedReader: utilizar o método readLine(), que nos permite ler linha a linha.

BufferedReader br = new BufferedReader(isr);

- representa o conjunto de caracteres. Este método nos retorna uma String, que representa a linha. Lembrando que, *BufferedReader* estende *Reader* mas o Reader por si só não possui o método readLine().

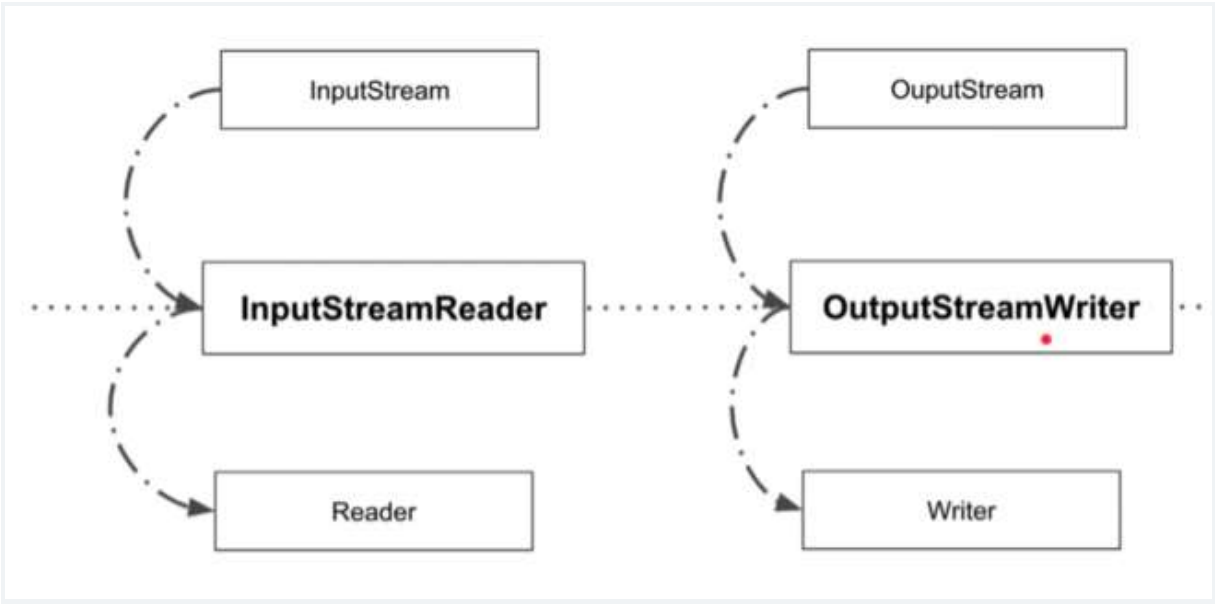
- Visualmente: BufferedReader > Reader > InputStream > lorem.txt.
  Esse padrão é um padrão de projeto chamado decorator, ou seja, um objeto está decorando a funcionalidade de outro, sucessivamente. Em geral, o java.io é repleto de padrões de projeto.

❖ Exception: O Java não é capaz de garantir que o desenvolvedor realmente inseriu o arquivo na raiz do projeto, por isso, o código está passível de falhas. Precisamos alertar sobre esta falha, e o modo pelo qual fazemos isso é a exceção do tipo checked.

Ao trabalharmos com java.io é necessário dominarmos dois tipos principais de exceção, a primeira é a FileNotFoundException e a segunda é a IOException.
Ao abrir a classe FileNotFoundException percebemos que ela é uma IOException, esta por sua vez, é uma exceção, já que estende Exception. Por isso, em vez de utilizarmos a exceção mais específica, utilizaremos o tipo mais genérico:

public static void main(String[] args) throws IOException.

- Escrita de arquivo: semelhante à leitura de arquivo. Utilizando o padrão de decorator.
  - OutputStream: que é análoga à InputStream. Filha: FileOutputStream (classe usada para escrever bytes num arquivo).
  - Writer: Filhas: OutputStreamWriter e BufferedWriter.
  - BufferedWriter.
- Atenção:Há classes que fazem a transição de um mundo para outro, como é o caso da *InputStreamReader*, que recebe um *InputStream* de bytes e o transforma em um Reader. Da mesma forma, temos o *OutputStreamWriter*, que faz o mesmo, só que para a escrita. Estas classes possuem padrões de projetos próprios do java.io.
  - InputStreamReader: transforma bytes em caracteres.
  - *OutputStreamWriter:* transforma caracteres em bytes .


InputStream
OuputStream
InputStreamReader
OutputStreamWriter
Reader
Writer