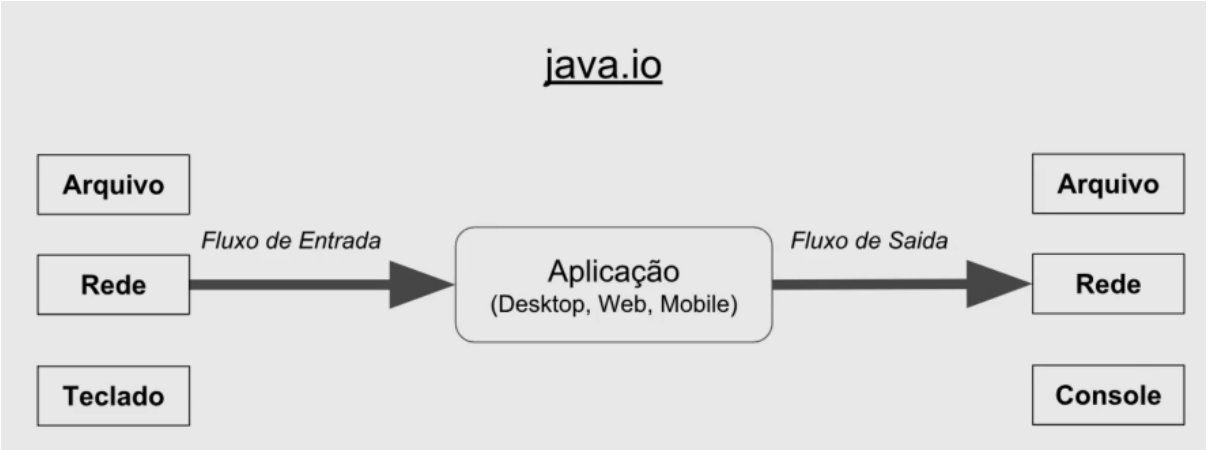
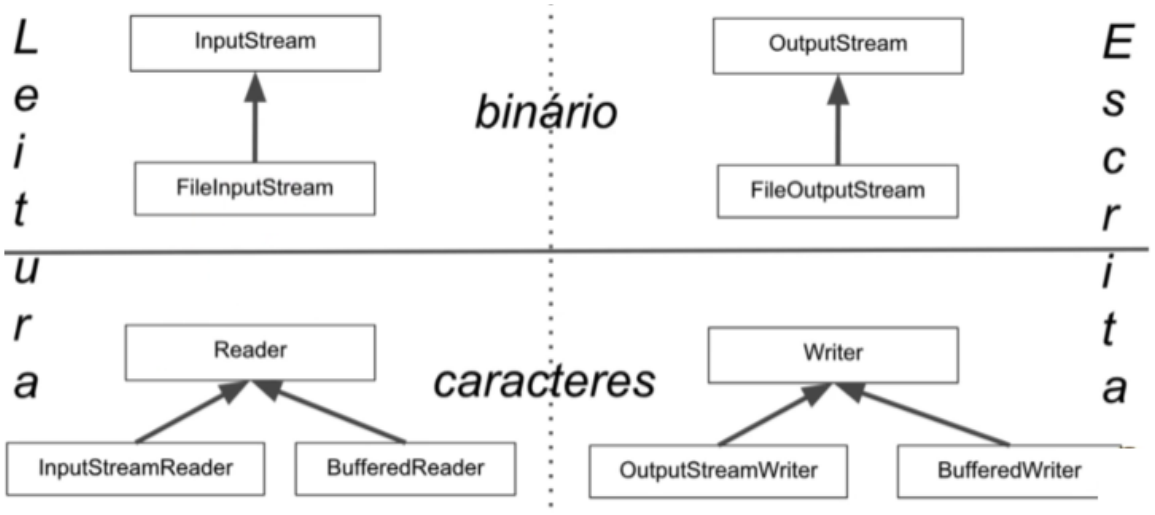


❖ STREAM e READER: Classes bases do Java.io e existem tanto para entrada quanto saída!!!

➢ 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).

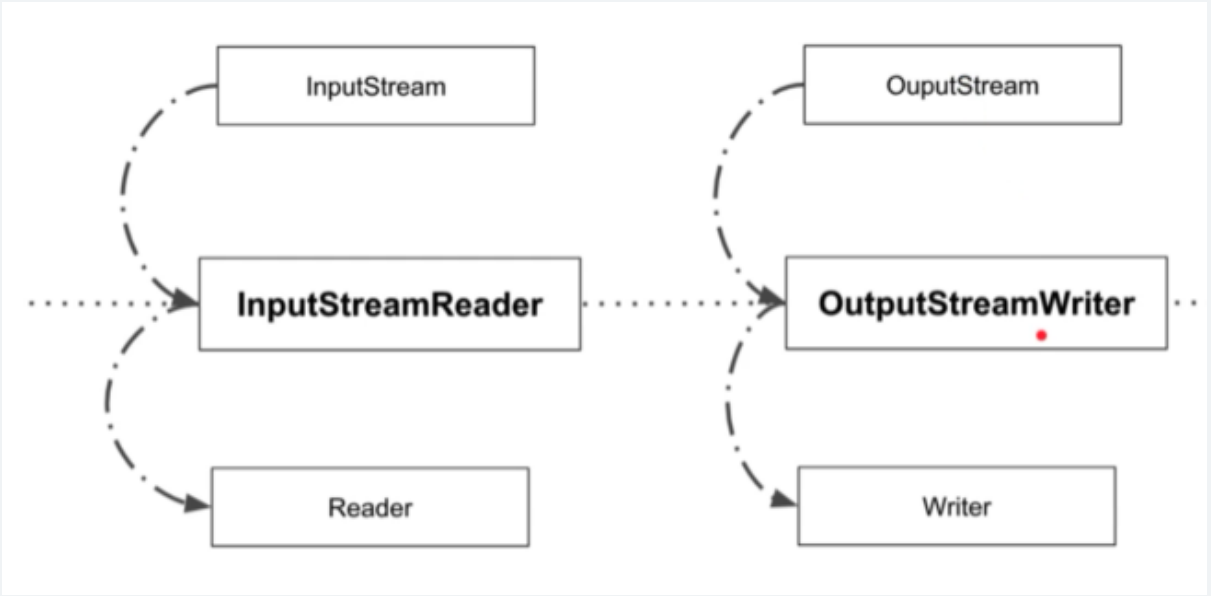
➢ Writer: Filhas: OutputStreamWriter e BufferedWriter.

➢ 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 .



❖ Outras formas de escrita:

❖ FileWriter: Classe usada para escrever caracteres. A classe FileWriter estende a classe OutputStreamWriter, que por sua vez estende a classe Writer. Então o FileWriter é um

OutputStreamWriter e é um Writer. Recomendável continuarmos utilizando o BufferedWriter.

```
FileWriter fw = new FileWriter("lorem2.txt");
BufferedWriter bw = new BufferedWriter(fw);
```

Simplificando e apenas passarmos o fw no seu construtor:

```
BufferedWriter bw = new BufferedWriter(new FileWriter("lorem2.txt"));
```

- PrintStream: Por meio dela é possível fazermos uma impressão para um fluxo binário. O PrintStream é uma classe de mais alto nível, que aceita uma grande variedade de construtores, como é o caso do new File(). Abrindo a classe PrintStream, vemos que ela existe desde a versão 1.0 do Java, enquanto as FileWriter e BufferedWriter entraram somente na versão 1.1. Ou seja, aqueles que desejavam trabalhar com caracteres desde o Java 1.0 utilizavam, necessariamente, a classe PrintStream. A partir disso, foram criadas ferramentas mais especializadas.

- PrintWriter: classe que funciona de forma análoga a PrintStream. Inicialmente existia somente o PrintStream, mas como depois surgiu o mundo de Writers, viu-se a necessidade de criar um PrintWriter, este que não precisa utilizar um Stream internamente.

❖ Encoding e Charsets

Há padrões de conversão de caracteres em binários, e um dos mais comuns está registrado na tabela ASCII (*American Standard Code for Information Interchange*). Temos a letra, e a ela há um número associado, a partir deste, é criada uma sequência binária. Por exemplo, a letra C é representada pelo número 67, e resulta na sequência 01000011.

Contudo, temos um problema pois esta tabela engloba todos os caracteres da língua inglesa, mas como sabemos há outros além destes. Por exemplo, aqueles que são acentuados. Há ainda outros alfabetos, com símbolos diferentes.

Para solucionar este problema, foram criadas as ***codepages***, único formato capaz de englobar a quantidade de informação correspondente ao número de línguas e caracteres existentes.

Para tentar unificar os padrões, foi criado o ***unicode***. Trata-se de uma tabela cujo objetivo é apresentar todos os caracteres existentes no mundo.

Ela também conta com um número associado a cada caractere (***codepoint***).

Contudo, o *unicode* não define a forma como as informações devem ser armazenadas no HD, isto é tarefa dos ***encodings***. Para representar o encoding existem as classes como é o caso dos "UTFs", como o UTF-8 e UTF-16, esta sigla significa "*Unicode Transformation Format*. Ela está vinculada desde o nascimento com a tabela de Unicode, para traduzir os *codepoints* para um formato binário.

Além do UTF há outros exemplos de *Encodings*, como o ASCII e o Windows 1252.

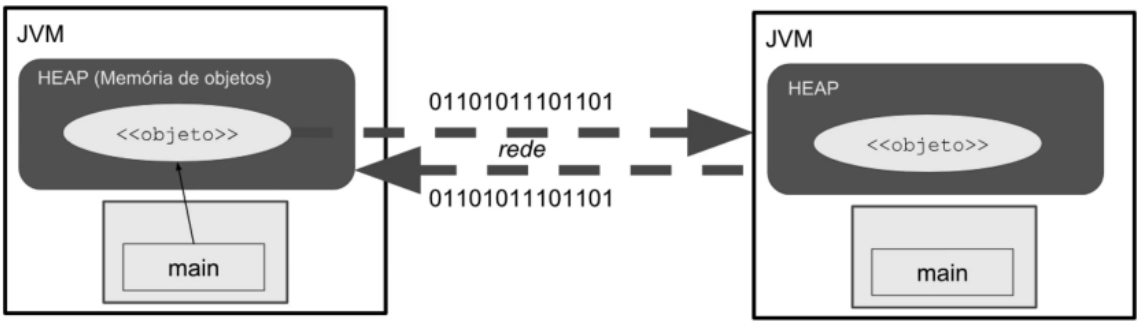
É importante termos a informação de qual é o charset padrão pois é ele que define como traduzir o codepoint em uma sequência de bits e bytes.

❖ Serialização

Dentro da JVM temos a memória de objetos (HEAP), e o main, que controla estes objetos. A serialização é a transformação do objeto Java, localizado na memória, em um fluxo de bits e bytes, e vice-versa.

- java.io.ObjectOutputStream = Objeto -> Bits e Bytes (Serialização)
- java.io.ObjectInputStream = Bits e Bytes -> Objeto (Desserialização)

Serialização/desserialização funciona em cascata e também com herança.



Qual o propósito de aprendermos isso? Uma situação que pode ocorrer é, por exemplo, gravarmos um objeto em um HD, e podemos recuperá-lo posteriormente. Isto está relacionado com a persistência.

O Java foi concebido com a intenção de funcionar em rede, ou seja, fazer com que duas máquinas virtuais se comuniquem em rede. Assim, é possível termos uma funcionalidade em uma JVM e transferi-la para outras, via rede, recebendo e enviando dados. No mundo Java, dados são objetos, portanto, este fluxo é realizado mediante a serialização.

- InvalidClassException: O erro InvalidClassException está relacionado com a serialização padrão Java. Ao gravarmos um objeto, estamos armazenando as informações que digitamos e, ao mesmo tempo, sem que isso dependa de nossa vontade, uma identificação da classe, um número que identifica a versão da classe. Se não especificamos um número de versão para a classe, o Java preenche automaticamente.

- serialVersionUID: O atributo serialVersionUID define a versão atual da classe e esse valor fica gravado na representação binária do objeto. Ao recuperar um arquivo, o Java acessa o número serial do arquivo, e o compara com o ID da classe, se forem iguais, ele continua, caso contrário, ele para. Ao trabalharmos com serialização, é uma boa prática inserirmos o serialVersionUID.
  - Caso não haja o número serial, a cada alteração na classe utilizada para criar um arquivo, causa uma nova versão, ou seja, um novo número gerado; Por este motivo, é boa prática forçar um número de versão, desta forma, as alterações - desde que compatíveis - ficarão armazenadas.
  - Cada alteração nos atributos, por exemplo a sua inclusão ou exclusão, merece uma alteração no número na versão.
  - Sempre quando serializamos o objeto, também será serializado o valor do serialVersionUID.