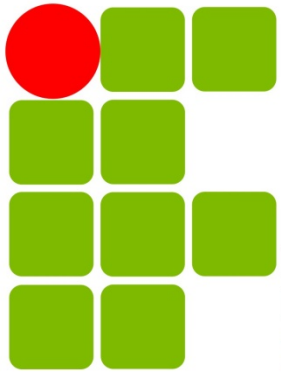


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA

Pacote java.io

Programação Orientado a Objetos

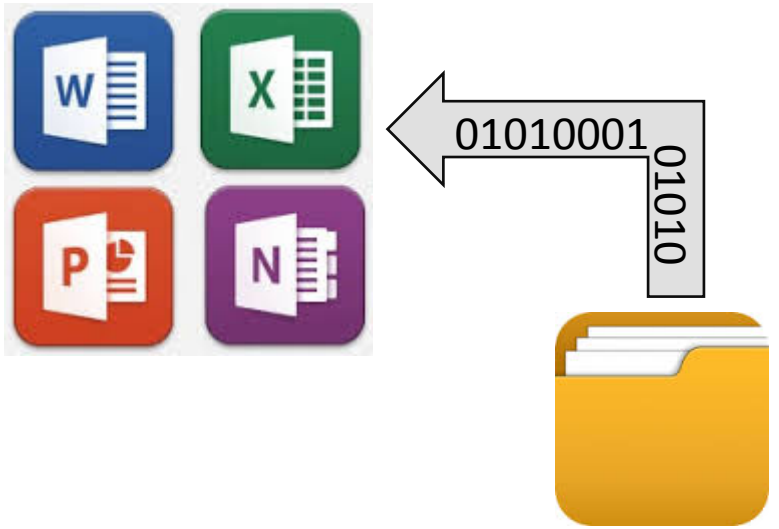
Profs. Danielle Chaves e Francisco Dantas

{cmedanielle, franciscodnn}@gmail.com

Campina Grande/PB – 2017

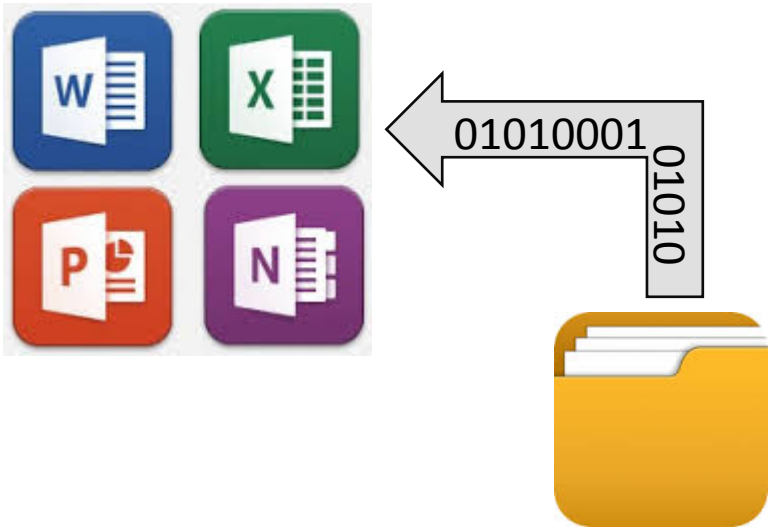
O que estudaremos nesta aula?

- Procedimentos para ler e escrever dados de/para a entrada e saída padrão;
- Como ler e escrever dados de/para arquivos;
- Como usar Scanner e PrintStream;
- Como ler e escrever objetos em arquivos.



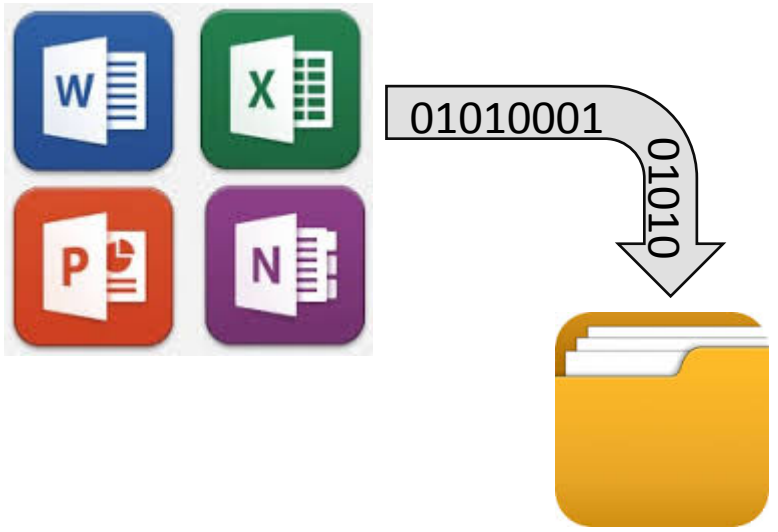
Fluxos de dados

Os aplicativos estão lendo (recebendo) o fluxo de dados do arquivo.



Fluxo de entrada de dados vindo do arquivo é chamado de ***InputStream***

Os aplicativos estão enviando (armazenando) o fluxo de dados no arquivo.



Fluxo de saída de dados para o arquivo é chamado de ***OutputStream***

Arquivos - Leitura

- Para ler dados de um **arquivo**, devemos usar a classe **FileInputStream**;
  - Se fossemos ler um **objeto** salvo em um arquivo, usaríamos uma classe chamada **ObjectInputStream**.

- Para se ler dados de um **arquivo**, é preciso saber **onde** o arquivo está no disco (pode ser usado caminho relativo ou absoluto);

Arquivos - Leitura

- Exemplo.

Usando caminho relativo!
OBS.: No eclipse, o arquivo estará na pasta raiz do projeto.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
      InputStream is = new FileInputStream("arquivo.txt");
      int b = is.read(); // Leitura de apenas um caractere do arquivo
   }
 }
```

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
      InputStream is = new FileInputStream("C:/arquivo.txt");
      int b = is.read(); // Leitura de apenas um caractere do arquivo
   }
 }
```

Usando caminho absoluto!

Arquivos - Leitura

- Ao trabalhar com arquivos, diversos métodos lançam uma exceção *IOException*:
  - Significa que algum erro em alguma operação entrada/saída pode ocorrer;
  - Seu programa **deve** tratar essa exceção (**IOException**), que é do **tipo checked (obrigatória)**:
    - Com o try{ ... } catch { ... }

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
      try{
         InputStream is = new FileInputStream("C:/arquivo.txt");
         int b = is.read(); // Leitura de apenas um caractere
      } catch(IOException ioe){
         System.out.println("Erro na leitura do arquivo!");
      }   }    }
```

Arquivos - Leitura

- Tratamento da exceção *IOException*.
  - Com o *try{ ... } catch { ... } finally { ... }*;
  - A cláusula *finally { ... }* é útil para inserção do código de fechamento do arquivo.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
      try{
         InputStream is = new FileInputStream("C:/arquivo.txt");
         int b = is.read(); // Leitura de apenas um caractere
      }catch(IOException ioe){
         System.out.println("Erro na leitura do arquivo!");
      }finally{
         is.close(); // Lembrar de sempre fechar o arquivo!
      }
   }
}
```

Arquivos - Leitura

- Se quisermos definir a codificação a ser lida de um arquivo (UTF_8, SO-8859-1, ASCII, etc), basta usar a classe *InputStreamReader*:
  - 1) Criar um objeto do tipo *FileInputStream* para o arquivo:
    - Definição do *fluxo de dados* de entrada no programa Java.
  - 2) Criar um objeto *InputStreamReader* que recebe como parâmetros o objeto do tipo *FileInputStream* e a codificação.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
     InputStream is = new FileInputStream("arquivo.txt");
     InputStreamReader isr = new InputStreamReader(is, StandardCharsets.UTF_8);
     int b = isr.read(); // Leitura de apenas um caractere do arquivo
     System.out.println((char) b);
   }
 }
```

Codificação UTF_8

Arquivos - Leitura

- Ler e imprimir todos os caracteres do arquivo:
  - A classe *InputStream* é abstrata, enquanto *InputStreamReader* é concreta.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
     InputStream is = new FileInputStream("arquivo.txt");
     InputStreamReader isr = new InputStreamReader(is, StandardCharsets.UTF_8);
     int caractere, contador = 0;
     // Percorrer todos os símbolos (chars) do arquivo
     // O final do arquivo é representado pelo -1
     while ((caractere = isr.read()) != -1){
        System.out.print((char) caractere); contador++;
     }
     // Fechar o arquivo
     isr.close();
     is.close();
   }
 }
```

Arquivos - Leitura

- Ler e imprimir todos os caracteres do arquivo.

```
class TestaEntrada {
   public static void main(String[] args) throws IOExce
      InputStream is = new FileInputStream("arquivo.txt"
      InputStreamReader isr = new InputStreamReader(is,
      int caractere, contador = 0;
      // Percorrer todos os símbolos (chars) do arquivo
      // O final do arquivo é representado pelo -1
      while ((caractere = isr.read()) != -1){
         System.out.print((char) caractere); contador++;
      }
      // Fechar o arquivo
      isr.close();
      is.close();
   }
}
```

Arquivo.txt

Disciplina: POO
Data da prova: ---
Assuntos: ---

Contador vai armazenar o quantitativo de caracteres do arquivo!

Variáveis	
contador	caractere
0	'D'
1	'i'
2	's'
3	'c'
4	'i'
...	

Arquivos - Leitura

- Apesar da classe abstrata *Reader* já ajudar no trabalho de manipulação de caracteres, ainda é difícil ler uma *String* (uma palavra inteira, por exemplo);
- A classe ***BufferedReader*** é um *Reader* que recebe outro *Reader* pelo construtor e concatena os diversos **chars** para formar uma *String* através do método *readLine*:

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
     InputStream is = new FileInputStream("arquivo.txt");
     InputStreamReader isr = new InputStreamReader(is);
     BufferedReader br = new BufferedReader(isr);
     String s = br.readLine();
   }
}
```

Arquivos - Leitura

- Leitura de arquivo com a classe *BufferedReader*.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
     InputStream is = new FileInputStream("arquivo.txt");
     InputStreamReader isr = new InputStreamReader(is, StandardCharsets.UTF_8);
     BufferedReader br = new BufferedReader(isr);
     String linha, contador = 0;
     // Percorrer todos as linhas do arquivo. Fim arquivo é retorna null
     while ((linha = br.readLine()) != null){
        System.out.print(linha); contador++;
     }
     // Fechar o arquivo
     br.close();
     isr.close();
     is.close();
   }
}
```

Arquivos - Leitura

- Modo mais simplificado de se ler do arquivo:
  - Uso da classe *BufferedReader*.

```
class TestaEntrada {
   public static void main(String[] args) throws IOExce
     InputStream is = new FileInputStream("arquivo.txt"
     InputStreamReader isr = new InputStreamReader(is,
     BufferedReader br = new BufferedReader(isr);
     String linha, contador = 0;
     // Percorrer todos as linhas do arquivo. Fim arqui
     while ((linha = br.readLine()) != null){
        System.out.print(linha); contador++;
     }
     // Fechar o arquivo
     br.close();
     isr.close();
     is.close();
   } }
```

Arquivo.txt

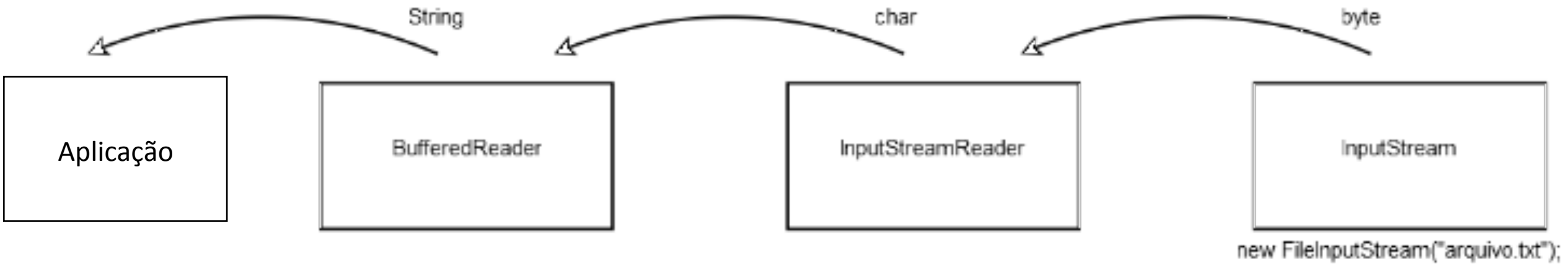
Disciplina: POO
Data da prova: ---
Assuntos: ---

Variáveis	
contador	linha
0	"Disciplina: POO"
1	"Data da pro... "
2	"Assuntos ..."

Contador vai armazenar o quantitativo de linhas do arquivo!

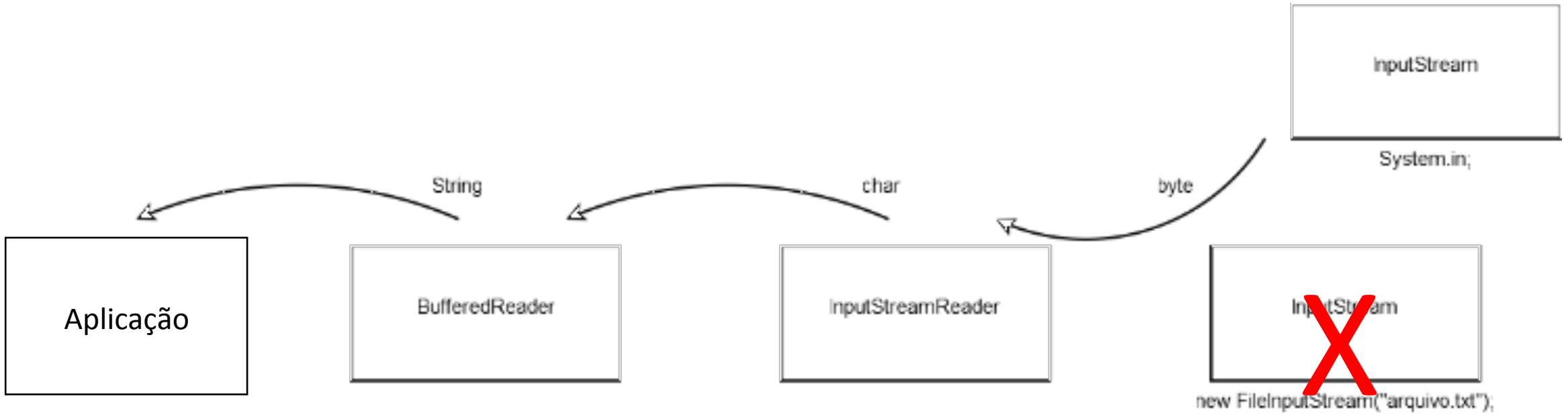
Arquivos - Leitura

- A imagem abaixo representa o encadeamento de classes utilizadas para leitura de um arquivo.



Leitura do Teclado

- Ao invés de ler de um arquivo, a leitura é feita em *System.in*.



Leitura do Teclado

- Para se ler do teclado, deve-se acessar a entrada padrão do sistema:
  - *System.in*.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
     InputStream is = System.in;
     InputStreamReader isr = new InputStreamReader(is);
     BufferedReader br = new BufferedReader(isr);
     String s = br.readLine();

     while (s != null) {
       System.out.println(s);
       s = br.readLine();
     }
   }
}
```

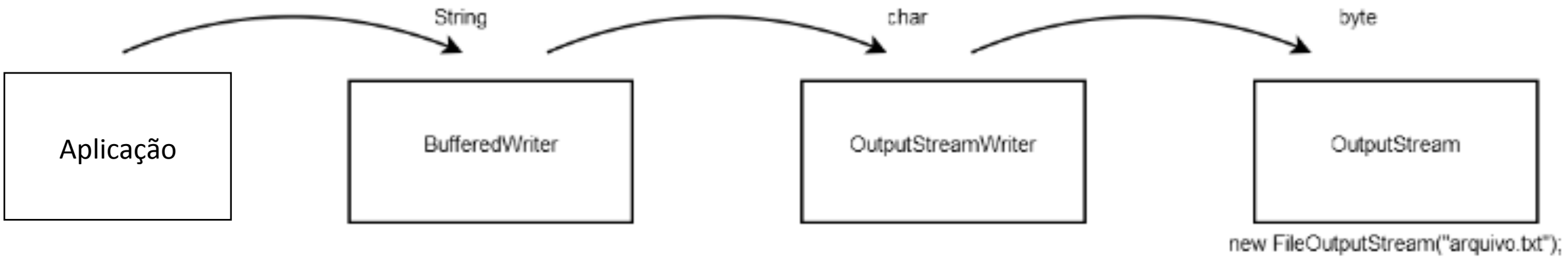
Leitura do Teclado

- Outra forma (mais simples) de se ler do teclado;
- *PrintStream* cria um fluxo de saída para o arquivo **"arquivo.txt"**.

```
class TestaEntrada {
   public static void main(String[] args) throws IOException {
      Scanner s = new Scanner(System.in);
      // Conexão de saída para o arquivo "arquivo.txt"
      PrintStream ps = new PrintStream("arquivo.txt");
      while (s.hasNextLine()) {
          ps.println(s.nextLine()); // Escreve no arquivo
      }
   }
}
```

Arquivos - Escrita

- Escrever em arquivo significa criar um ***fluxo de saída de dados*** da aplicação para um arquivo;
- Para criar um ***fluxo de saída de dados*** na aplicação, usamos um *OutputStream*:
  - Lembre-se: para criar um *fluxo de entrada de dados*, usamos um *InputStream*.



Arquivos - Escrita

- Passos para se escrever em um arquivo:
  - 1) Cria-se um *fluxo de saída* para um arquivo;
  - 2) Define-se a codificação a ser utilizada na escrita, pelo *Writer*;
  - 3) Cria-se o *buffer* e escreve-se o texto de fato em um *fluxo de saída*.

```
class TestaEscrita {
   public static void main(String[] args) throws IOException {
     OutputStream os = new FileOutputStream("saida.txt"); // Passo 1
     OutputStreamWriter osw = new OutputStreamWriter(os); // Passo 2
     BufferedWriter bw = new BufferedWriter(osw); // Passo 3

     bw.write("Disciplina de POO"); // Passo 3
     osw.close(); // Lembrar de fechar o arquivo!
     os.close();
   }
}
```

Arquivos - Escrita

- Maneira mais apropriada e elegante de se escrever em um arquivo.
  - Tratar a exceção ***IOException***, ao invés de lançá-la.

```
class TestaEscrita {
   public static void main(String[] args) throws IOException {
     try{
         OutputStream os = new FileOutputStream("saida.txt");
         OutputStreamWriter osw = new OutputStreamWriter(os);
         BufferedWriter bw = new BufferedWriter(osw);
         bw.write("Disciplina de POO");
         bw.newLine(); // Insere uma quebra de linha no arquivo!
      } catch(IOException ioe){
         ioe.printStackTrace();
         System.out.println("Erro na escrita!");
      } finally{
         bw.close(); // Lembrar de fechar o arquivo - leitura e escrita!
         osw.close(); os.close();
     }
   } }
```

Tratar a exceção IOException é boa prática!

Exercícios

1. **Ajuste o código apresentado no slide 11, para que o tratamento da exceção seja realizada apropriadamente (ao invés de lançar a exceção para o programa principal);**

2. **Utilizando a classe *BufferedWriter*, crie um programa para escrever seu nome na primeira linha de um arquivo, sua idade na segunda linha e seu livro favorito na terceira linha;**

3. **Utilizando a classe *BufferedReader*, crie um programa para ler o conteúdo do arquivo (questão 2) e imprima na console do programa Java.**

Dicas de Leitura & Referências Utilizadas

- Apostila Java e Orientação a Objetos – Capítulo 15 Pacote java.io
- Javadoc do pacote java.io