



C125 – Programação Orientada a Objetos com  
Java

## O Pacote java.io

Prof. Phyllipe Lima  
phyllipe@inatel.br



1



## Agenda



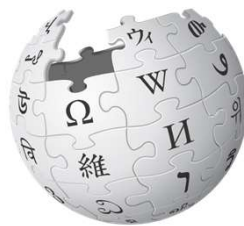
- ☕ Ler **bytes**, caracteres e **String** de arquivos
- ☕ Escrever **bytes** e **Strings** em arquivos
- ☕ Exercícios

2

## API (*Application Programming Interface*)



☕ Uma API é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software



WIKIPÉDIA  
A enciclopédia livre

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

3

3

## Somos Todos Objetos!



☕ Assim como todo o resto das bibliotecas (APIs) em Java, a parte de controle de entrada e saída de dados (conhecido como io – **input/output**) é orientada a objetos e usa conceitos que vimos ao longo do curso como **interfaces**, **classes abstratas** e **polimorfismo**.

☕ Utilizando os recursos OO, conseguimos criar programa genéricos para leitura de dados (com a classe **InputStream**) e escrita de dados (com a classe **OutputStream**) sem nos preocuparmos onde os dados serão lidos/escritos, como por exemplo: arquivos, banco de dados, conexão remota com **sockets** (através da rede) e até mesmo à entrada e saída padrão de um programa (normalmente o teclado e o console, como já vimos no início do curso).

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

4

4

## Lendo Arquivos

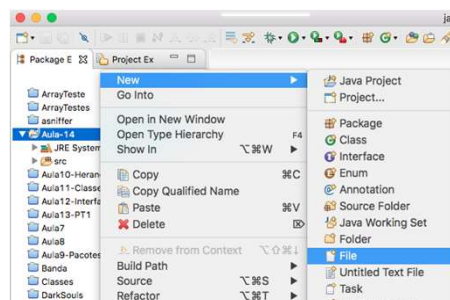


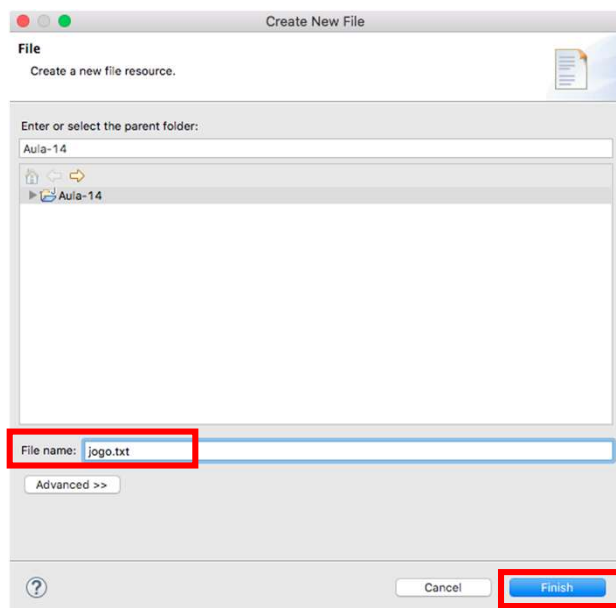
- ☞ Antes de lermos um arquivo, precisamos **ter** um arquivo. Podemos criar arquivos em qualquer diretório. Para facilitar nosso aprendizado, faremos isso na raiz onde está o projeto que estamos trabalhando no Eclipse.
- ☞ Podemos fazer isso, inclusive, dentro do próprio Eclipse, sem utilizar o explorador de arquivos do sistema operacional.
- ☞ Vamos criar um arquivo chamado “jogo.txt”

## Arquivos



- ☞ Considere que já existe um projeto dentro do Eclipse e faça as seguintes operações (nesse exemplo estou no projeto Aula-14)
- ☞ Botão direito no projeto -> New (Novo) -> File (Arquivo)





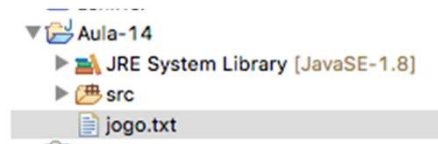
☕ Dê o nome jogo.txt



7

## Arquivos

☕ Observe que foi criado, na raiz, um arquivo chamado “jogo.txt”



☕ Você pode utilizar o explorador de arquivos do seu sistema operacional para verificar que, de fato, está criado esse arquivo



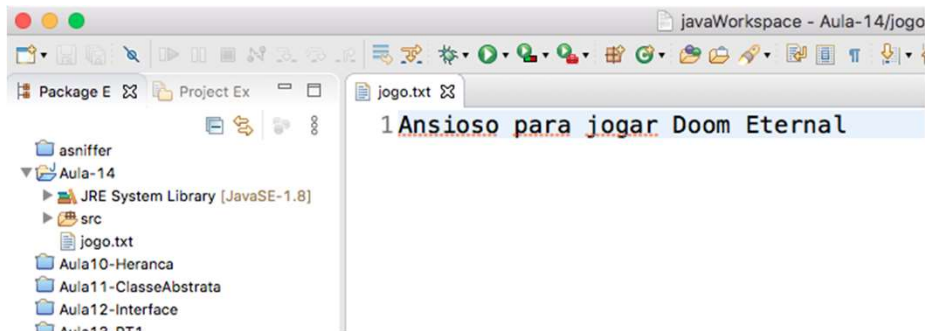
8



## O que vamos ler?



☕ Com o arquivo aberto, escreva algo que lhe agrade!



## Como ler?



☕ O Java, através de uma API, nos fornece uma implementação concreta para ler os **bytes** de um arquivo. É importante lembrarmos que dentro do computador o que temos são **bytes** salvos na memória. A classe que usaremos é a **FileInputStream**.

☕ Para ler o arquivo precisamos passar **onde** ele está salvo. Fazemos isso no próprio construtor da classe.



## Vida Longa ao Polimorfismo

```

1 package br.inatel.cdg;
2
3 import java.io.FileInputStream;
4 import java.io.InputStream;
5
6 public class Main {
7
8     public static void main(String[] args) {
9
10         InputStream input = new FileInputStream();
11     }
12

```

Observe os **imports**. Temos o **java.io.FileInputStream** que é nossa classe concreta. Temos também **java.io.InputStream** que é uma classe abstrata. No código da linha 10, atribuímos uma instância de **FileInputStream** para uma variável do tipo **InputStream**. Estamos utilizando o polimorfismo. Isso funciona por que **FileInputStream** herda de **InputStream**.

A vantagem é que poderemos passar essa instância para qualquer método que receba **InputStream**. E esses métodos podem também trabalhar com qualquer classe filha de **InputStream**. Assim, o método fica genérico.



## Onde Está o Arquivo?

Você deve ter observado que o Eclipse está apontando um erro. É porque precisamos passar **no construtor** onde está o arquivo que queremos abrir. É um parâmetro obrigatório. O próprio Eclipse nos ajuda a identificar esse erro. Coloque o cursor do mouse do lado esquerdo da linha 10

```

6 public class Main {
7
8     public static void main(String[] args) {
9
10        The constructor FileInputStream() is undefined. = new FileInputStream();
11    }
12

```

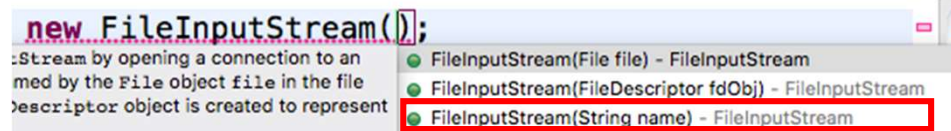
Em outras palavras “O construtor sem parâmetros não foi definido, passe alguma coisa”.

## Onde Está o Arquivo?



Clique dentro do construtor e dê um Ctrl + Espaço (*intelliSense* do Eclipse). Ele lhe apresentará as opções.

Veja que podemos passar alguns tipos de parâmetros, incluindo uma String.



Essa parâmetro é o caminho completo do arquivo. Como criamos dentro da raiz do projeto do Eclipse, passamos apenas "jogo.txt".

Se esse arquivo estivesse em outro lugar, como na área de trabalho, deveríamos passar o caminho completo. Exemplo: "/Users/phillima/Desktop/jogo.txt". No seu computador esse caminho deve ser outro.

Fique atento como seu sistema operacional separa caminhos de diretórios. Windows e UNIX possuem sistemas diferentes.

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

13

13

## Onde Está o Arquivo?



Vamos criar uma variável do tipo String para salvar o nome do arquivo e passar como parâmetro para o construtor. Poderíamos fazer diretamente. Mas é boa prática utilizarmos variáveis para esse propósito.

```

6 public class Main {
7
8     public static void main(String[] args) {
9
10         String arquivo = "jogo.txt";
11         InputStream input = new FileInputStream(arquivo);
12     }
13
14 }
```

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

14

14



## Exceção a Regra

☞ O Eclipse ainda está acusando um erro. Sabemos que, da aula de Exceções, precisamos tratar uma **checked exception** ao ler arquivos.

☞ Podemos fazer através de um **try/catch** ou um **throws**. Dado que estamos no método **main()** vamos utilizar o bloco **try/catch**.

☞ Utilize os recursos do Eclipse para facilitar.

```

9 public static void main(String[] args) {
10
11     String arquivo = "jogo.txt";
12     try {
13         InputStream input = new FileInputStream(arquivo);
14     } catch (FileNotFoundException e) {
15         System.out.println("Arquivo " + arquivo + " não encontrado");
16     }
17 }

```

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

15

15



## Lendo Caracteres

☞ Ao executarmos esse programa nada acontece. Mas também não tivemos Exceção lançada. Significa que o arquivo foi encontrado pela classe **FileInputStream**.

☞ Para lermos um caractere escrito dentro desse arquivo precisamos traduzir os **bytes** que estão salvos seguindo algum **encoding (codificação)** de caractere. O Java possui um **decoder** de caractere, implementado na classe **InputStreamReader**. O construtor dessa classe recebe uma instância do tipo **InputStream**. Ele pode receber também qual tipo de **encoding**. Se não passarmos nada, o padrão é UTF-8.

☞ A classe **InputStreamReader** é filha da classe abstrata **Reader**, que por sua vez possui outras classes filhas.

```

public static void main(String[] args) {
    String arquivo = "jogo.txt";
    try {
        InputStream input = new FileInputStream(arquivo);
        Reader inputSR = new InputStreamReader(input);
    } catch (FileNotFoundException e) {
        System.out.println("Arquivo " + arquivo + " não encontrado");
    }
}

```

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

16

16



## Queremos Ler Palavras



Da forma como o programa se encontra, conseguimos ler **caracteres**. É possível ler um por um e concatena-los manualmente em uma String. Mas o Java já fornece um recurso para essa tarefa.

Vamos utilizar a classe **BufferedReader** e o método **readLine()**.

Esse método irá ler uma linha do arquivo, e podemos criar um **loop** para ler todas as linhas do arquivo, até que não tenha mais nada a ser lido. Sabemos que o arquivo chegou ao fim quando **readLine()** retornar **null**

Observe que ao fazermos a leitura da linha, uma outra exceção pode ser lançada, chamada **IOException**. Então devemos coloca-la também dentro do **catch**, ou fazer outro bloco **catch** separado.

Precisamos também fechar o arquivo após utiliza-lo. Fazemos isso com a última instância de quem usou o arquivo, ou seja, o **BufferedReader**.

```
public static void main(String[] args) {
    String arquivo = "jogo.txt";
    try {
        InputStream input = new FileInputStream(arquivo);
        Reader inputSR = new InputStreamReader(input);
        BufferedReader leitor = new BufferedReader(inputSR);

        String texto = leitor.readLine();

        while(texto != null) {
            System.out.println(texto);
            texto = leitor.readLine();
        }

    } catch (FileNotFoundException e) {
        System.out.println("Arquivo " + arquivo + " não encontrado");
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

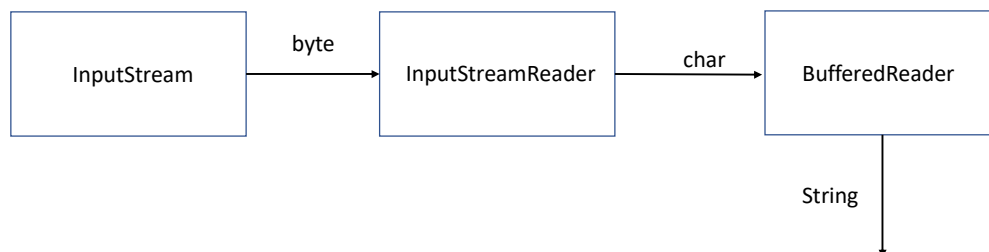




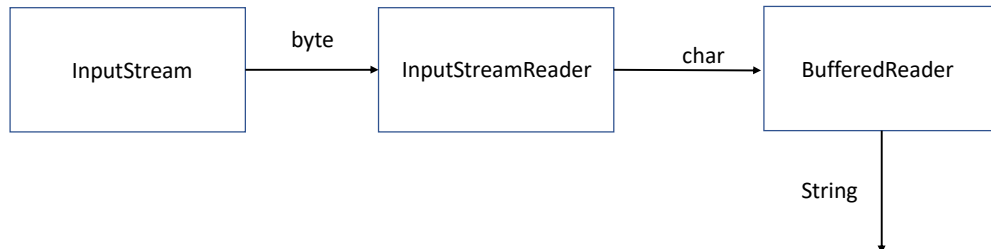
```
public static void main(String[] args) {  
    String arquivo = "jogo.txt";  
    try {  
        InputStream input = new FileInputStream(arquivo);  
        Reader inputSR = new InputStreamReader(input);  
        BufferedReader leitor = new BufferedReader(inputSR);  
  
        String texto = leitor.readLine();  
  
        while(texto != null) {  
            System.out.println(texto);  
            texto = leitor.readLine();  
        }  
  
        leitor.close(); //Para fechar o arquivo  
    } catch (FileNotFoundException e) {  
        System.out.println("Arquivo " + arquivo + " não encontrado");  
    } catch (IOException e) {  
        System.out.println(e.getMessage());  
    }  
}
```



## Fluxo de Leitura

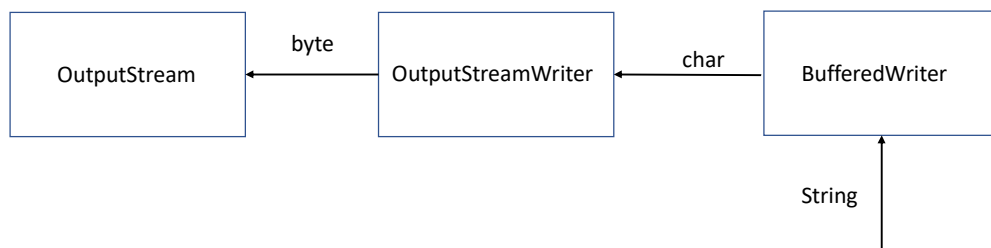


## Fluxo de Leitura



- Como visto, para ler Strings em Java seguimos o fluxo mostrado acima
- Primeiro fazemos a leitura de bytes, depois passamos para caracteres, e finalmente para String. Em cada etapa existe uma classe responsável.

## Fluxo de Escrita



- Já podemos imaginar que o fluxo de saída segue uma lógica parecida.
- Primeiro temos a String, depois convertemos para caracteres, e finalmente para bytes que serão escritos no arquivo.

```

10= public static void main(String[] args) {
11     try {
12         OutputStream arquivo = new FileOutputStream("arquivo.txt");
13         OutputStreamWriter writer = new OutputStreamWriter(arquivo);
14         BufferedWriter bWriter = new BufferedWriter(writer);
15
16         bWriter.write("Eu ainda quero jogar Doom Eternal!");
17
18         bWriter.close();
19
20     } catch (FileNotFoundException e) {
21         e.printStackTrace();
22     } catch (IOException e1) {
23         e1.getMessage();
24     }

```



🔥 O arquivo é criado na linha 12. Como passamos apenas o nome, ele será criado na raiz do projeto do Eclipse. Na linha 13 criamos um instância capaz de escrever caracteres. E na linha 14 criamos o **BufferedWriter** que é capaz de escrever **Strings**.

23

🔥 Na linha 16, chamamos o método **write()** na instância **bWriter**. E conseguimos passar **Strings** nesse método.

🔥 Na linha 18 precisamos chamar o **close()** para fechar o arquivo com segurança e ter as alterações salvas corretamente.



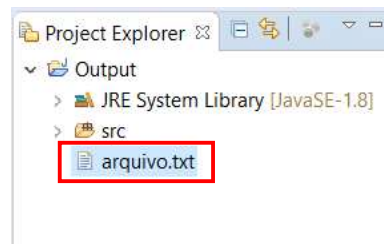
```

10= public static void main(String[] args) {
11     try {
12         OutputStream arquivo = new FileOutputStream("arquivo.txt");
13         OutputStreamWriter writer = new OutputStreamWriter(arquivo);
14         BufferedWriter bWriter = new BufferedWriter(writer);
15
16         bWriter.write("Eu ainda quero jogar Doom Eternal!");
17
18         bWriter.close();
19
20     } catch (FileNotFoundException e) {
21         e.printStackTrace();
22     } catch (IOException e1) {
23         e1.getMessage();
24     }

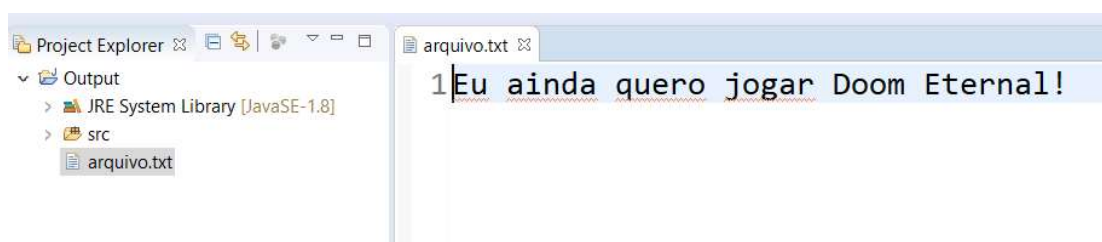
```

24

- ☕ O arquivo criado, “arquivo.txt”, se encontra na raiz do projeto do Eclipse.
- ☕ É possível encontrar esse arquivo buscando diretamente no explorador de arquivos do seu sistema operacional.
- ☕ Ele também pode ser visualizado dentro do próprio Eclipse.
- ☕ Aperte F5 para fazer um **refresh** dos arquivos. Ou clique com o botão direito no projeto e clique na opção **refresh**.
- ☕ O “arquivo.txt” estará visível na hierarquia do **Project Explorer**.



- ☕ O seu conteúdo também poderá ser visualizado dentro do eclipse, ao abrir o arquivo.



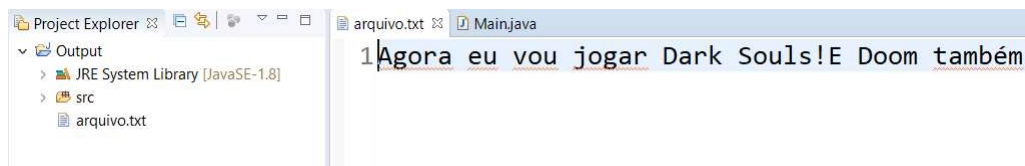
Outra opção é usar o método **append()**. Que é mais moderno e tem algumas vantagens. Entre elas ele pode ser encadeado, e também pode receber um **StringBuilder** (uma classe capaz de concatenar Strings em Java). Abaixo um exemplo com encadeamento.



Isso funciona pois o **append()** retorna uma instância de **Writer**

```
bWriter.append("Agora eu vou jogar Dark Souls!")
        .append("E Doom também");
```

Resultado



C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

27

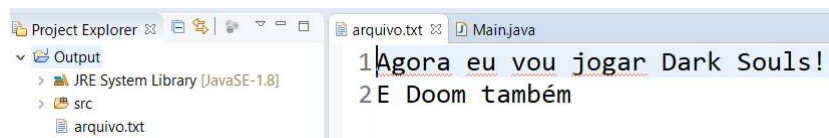
27

Para escrever em uma linha nova podemos inserir manualmente o meta-caractere de linha nova “\n”.



```
bWriter.append("Agora eu vou jogar Dark Souls!\n")
        .append("E Doom também");
```

Resultado



C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

28

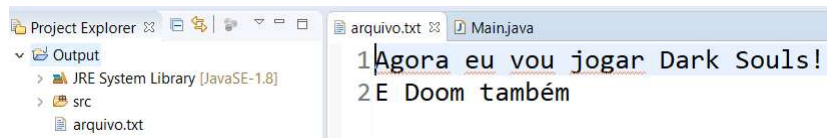
28

☕ Existe também o método ***newLine()*** que pode ser chamado por uma instância de ***BufferedWriter***. Mas assim não conseguimos fazer o encadeamento com o ***append***.



```
bWriter.append("Agora eu vou jogar Dark Souls!");
bWriter.newLine();
bWriter.append("E Doom também");
```

☕ Resultado




☕ Em todos os casos anteriores, estamos abrindo o arquivo e apagando tudo que se encontra lá antes de escrever. Se desejarmos continuar escrevendo a partir do que já se encontra, devemos passar o parâmetro ***true*** para o construtor de ***FileOutputStream***. É o segundo parâmetro. Quando não passamos nada, o Java considera ***false*** como padrão.



```
OutputStream arquivo = new FileOutputStream("arquivo.txt", true);
OutputStreamWriter writer = new OutputStreamWriter(arquivo);
BufferedWriter bWriter = new BufferedWriter(writer);


bWriter.append("Agora eu vou jogar Dark Souls!");
bWriter.newLine();
bWriter.append("E Doom também");
```





 Ao executar o programa temos o seguinte resultado



```
1 Agora eu vou jogar Dark Souls!|
2 E Doom também Agora eu vou jogar Dark Souls!
3 E Doom também
```

 O que está correto. Pois o “arquivo.txt” já existia. E seu conteúdo era:  
“Agora eu vou jogar Dark Souls!  
E Doom também”


 Como agora estamos no modo para **continuar escrevendo**. O programa começou a escrever no fim do arquivo. Não necessariamente em uma linha nova. P

 Podemos modificar o código para sempre escrevermos uma linha nova no fim.



```
OutputStream arquivo = new FileOutputStream("arquivo.txt", true);
OutputStreamWriter writer = new OutputStreamWriter(arquivo);
BufferedWriter bWriter = new BufferedWriter(writer);

bWriter.append("Agora eu vou jogar Dark Souls!");
bWriter.newLine();
bWriter.append("E Doom também\n");
```

 É manipular o “\n” de forma adequada, para obter os resultados desejados.



## Do Teclado Para Um Arquivo



- ☞ Já sabemos utilizar a classe **Scanner** para fazer leituras do teclado, como vimos no início do curso.
- ☞ E já sabemos também escrever em arquivos usando as classes **BufferedReader, OutputStream e FileOutputStream**.
- ☞ Podemos também utilizar a classe **PrintStream** que é capaz de escrever Strings **diretamente** em um arquivo. Sem a necessidade de conversão de String para caractere para **bytes**. A diferença é que perdemos o controle sobre o **encoding** utilizado.
- ☞ Considere o próximo exemplo que faz leitura do teclado, e escreve diretamente em um arquivo as Strings.

```
Scanner entrada = new Scanner(System.in);
PrintStream ps = new PrintStream("arquivo2.txt");

while(entrada.hasNextLine()) {
    ps.print(entrada.nextLine());
}

ps.close();
```



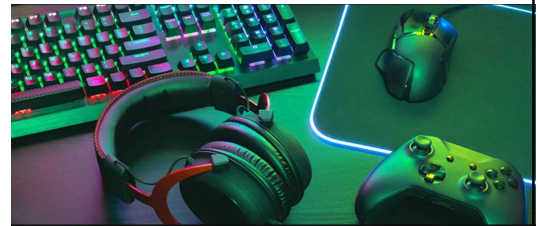
- ☞ Observe como fica muito mais simples abrir um arquivo e escrever uma **string** nele. Enquanto o método **hasNextLine** retornar **true** ele irá continuar escrevendo. O método **print()** da classe **Printstream** é responsável por escrever a string. Quem está lendo a string do teclado é **nextLine()**.
- ☞ Para interromper a execução é necessário passar o comando de fim de arquivo EOF. No Windows é Ctrl+Z e em máquinas UNIX é Ctrl+D

## Exercício 1 - Desafio



- 🔥 Crie um arquivo chamado "jogos.txt"
- 🔥 Usando a classe **BufferedWriter** escreva em cada linha o nome do jogo + ";" + nome estúdio desenvolvedor. Podem ser nomes fictícios. Exemplo do arquivo:

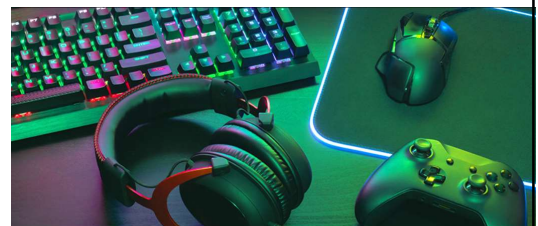
```
Dark Souls ; From Software  
Doom ; ID Software  
World of Warcraft ; Blizzard  
Zombicide ; CMNO|
```



## Exercício 1 (Cont.)



- 🔥 Depois de criado o arquivo, escreva um programa para ler o arquivo e criar dois arrays. Um array irá conter o nome dos jogos, e o outro o estúdio desenvolvedor. Depois imprima ambos esses arrays.
- 🔥 Dicas: Você precisará manipular a classe String para conseguir quebrar uma linha usando um caractere específico como separador. Nesse exemplo é o ";".



## Exercício 2 – Desafio



- ☕ Crie um programa que peça para o usuário entrar com uma frase qualquer.
- ☕ Em seguida utilize a Cifra de César para criptografar essa frase.
- ☕ Salve a frase criptografada em um arquivo
- ☕ Crie um código capaz de ler a string do arquivo e fazer a decifragem.
- ☕ Envie para um colega de sala o arquivo com o texto criptografado. Ele deverá Utilizar o programa para ver a mensagem Original. Vocês devem compartilhar a chave. Nesse caso o deslocamento das letras.



## Vídeo Aula



☕ <https://youtu.be/6see0bj5HrI>

☕ OBS (2020/1)

## Resolução dos Exercícios





<https://github.com/phillima-inatel/C125>



## Material Complementar



 Capítulo 16 da apostila FJ-11

 Pacote Java.io

 Até o item 16.4



C125 – Programação Orientada a Objetos com  
Java

## O Pacote java.io

Prof. Phyllipe Lima  
phyllipe@inatel.br

