



Tópicos avançados em Programação

Manipulação de arquivos

<http://dl.dropbox.com/u/3025380/prog2/aula5.pdf>

flavio.cec@unisul.br

Manipulação de arquivos

- Diariamente utilizamos arquivos em nossos computadores, para armazenar e consultados dados.
- Este é um recurso bastante importante em programação, pois permite que dados sejam persistidos ao final da execução de um programa.



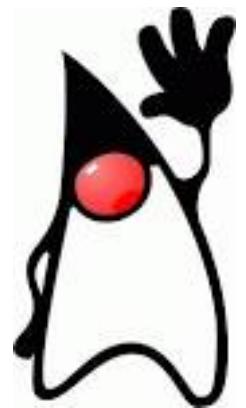
Manipulação de arquivos

- Até então mantínhamos os dados de nossas aplicações em memória, de modo que quando a aplicação fosse encerrada os dados eram perdidos.
- A persistência de dados em arquivo, pode ser um alternativa para o problema apresentado.





Como é possível ler arquivos em
Java?



Pacote *java.io*

- O java trata a entrada e saída como fluxos de dados (os tão chamados *Streams*);
- As classes ligadas a io estão nos pacotes `java.io` e `java.nio`;
- Instâncias da classe *java.io.File* representam caminhos (paths) para possíveis locais no sistema operacional.

```
//Declarando e instanciando um arquivo  
File arquivo = new File("conteudo/arquivoEntrada.txt");
```

Classe *File*

- Essa classe está diretamente ligada à um caminho;
- Esse caminho pode existir ou não;
- Através deste objeto é possível verificar propriedades de um arquivo ou pasta.
 - Mas ele não interage com o seu conteúdo.
- É possível também criar arquivos e pastas, mas não escrever dentro do mesmo.

Classe *File*

Só é possível acessar este arquivo, porque a pasta conteúdo, foi adicionada no classloader de execução.


```
//Declarando e instanciando um arquivo
File arquivo = new File("conteudo/arquivoEntrada.txt");

//Verifica se o arquivo existe
boolean arquivoExiste = arquivo.exists();

System.out.println("Arquivo existe: "+arquivo.exists());

//Se o arquivo existir é apresentado o seu caminho completo
if(arquivoExiste) {
    System.out.println(arquivo.getAbsolutePath());
}
```

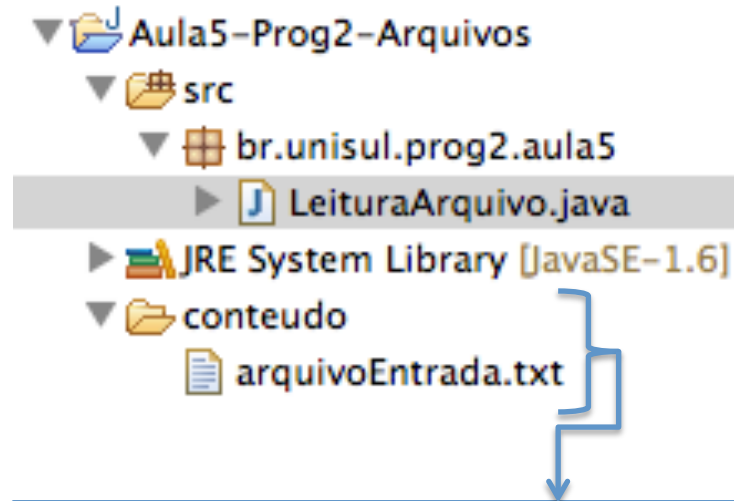
CONSOLE



The screenshot shows an IDE window with tabs for 'Problems', 'Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a Java application named 'LeituraArquivo'. The output shows the file path and confirms the file's existence.

```
<terminated> LeituraArquivo [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/jav
Arquivo existe: true
/Users/flavioceci/Documents/workspace/wsunisul/Aula5-Prog2-Arquivos/conteudo/arquivoEntrada.txt
```

ClassPath



Para configurar as pastas no classPath com visibilidade durante a execução, deve-se mapear as pastas na invocação do método main.

Como a pasta está interna ao projeto do eclipse, a IDE já adiciona a pasta (e seus arquivos) no ClassPath que ganha visibilidade dentro da aplicação.



Linux ou Mac: `java -cp './bin:./conteudo' br.unisul.prog2.aula5.LeituraArquivo`

OU

Windows: `java -classpath ".\bin;./conteudo" br.unisul.prog2.aula5.LeituraArquivo`

Criando subpasta

```
//Declarando e instanciando uma pasta
File pastaRaiz = new File("conteudo/");

//Verifica se a pasta existe
boolean pastaExiste = pastaRaiz.exists();

System.out.println("Arquivo existe: "+pastaRaiz.exists());

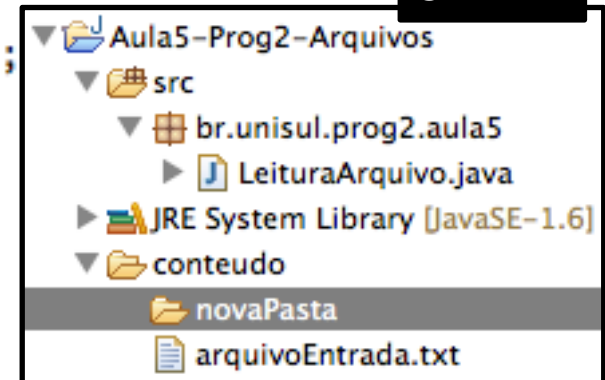
//Se a pasta existir é criada uma subpasta
if(pastaExiste) {
    File novaPasta = new File("conteudo/novaPasta/");

    //Caso a subpasta não exista ela é criada
    if(novaPasta.exists() == false) {
        //Criação da subpasta
        novaPasta.mkdir();

    }

    System.out.println(novaPasta.getAbsolutePath());
}
```

SAÍDA



Criando arquivo

```
//Declarando e instanciando uma pasta
File pastaRaiz = new File("conteudo/");

//Verifica se a pasta existe
boolean pastaExiste = pastaRaiz.exists();

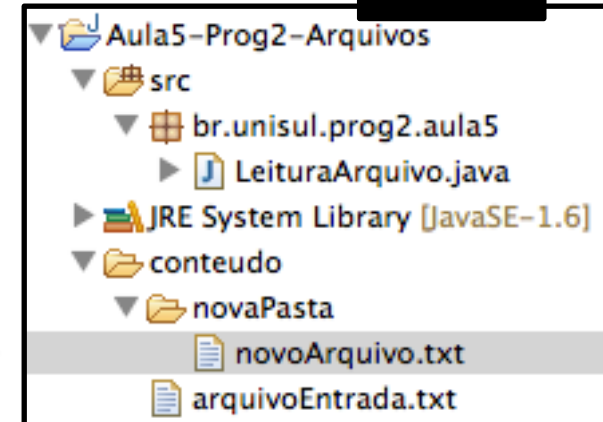
System.out.println("Arquivo existe: "+pastaRaiz.exists());

//Se a pasta existir é criada uma subpasta
if(pastaExiste) {
    File novaPasta = new File("conteudo/novaPasta/");

    //Caso a subpasta não exista ela é criada
    if(novaPasta.exists() == false) {
        //Criação da subpasta
        novaPasta.mkdir();
    } else {
        File novoArquivo = new File("conteudo/novaPasta/novoArquivo.txt");
        //Criando novo arquivo (lança IOException)
        novoArquivo.createNewFile();
    }

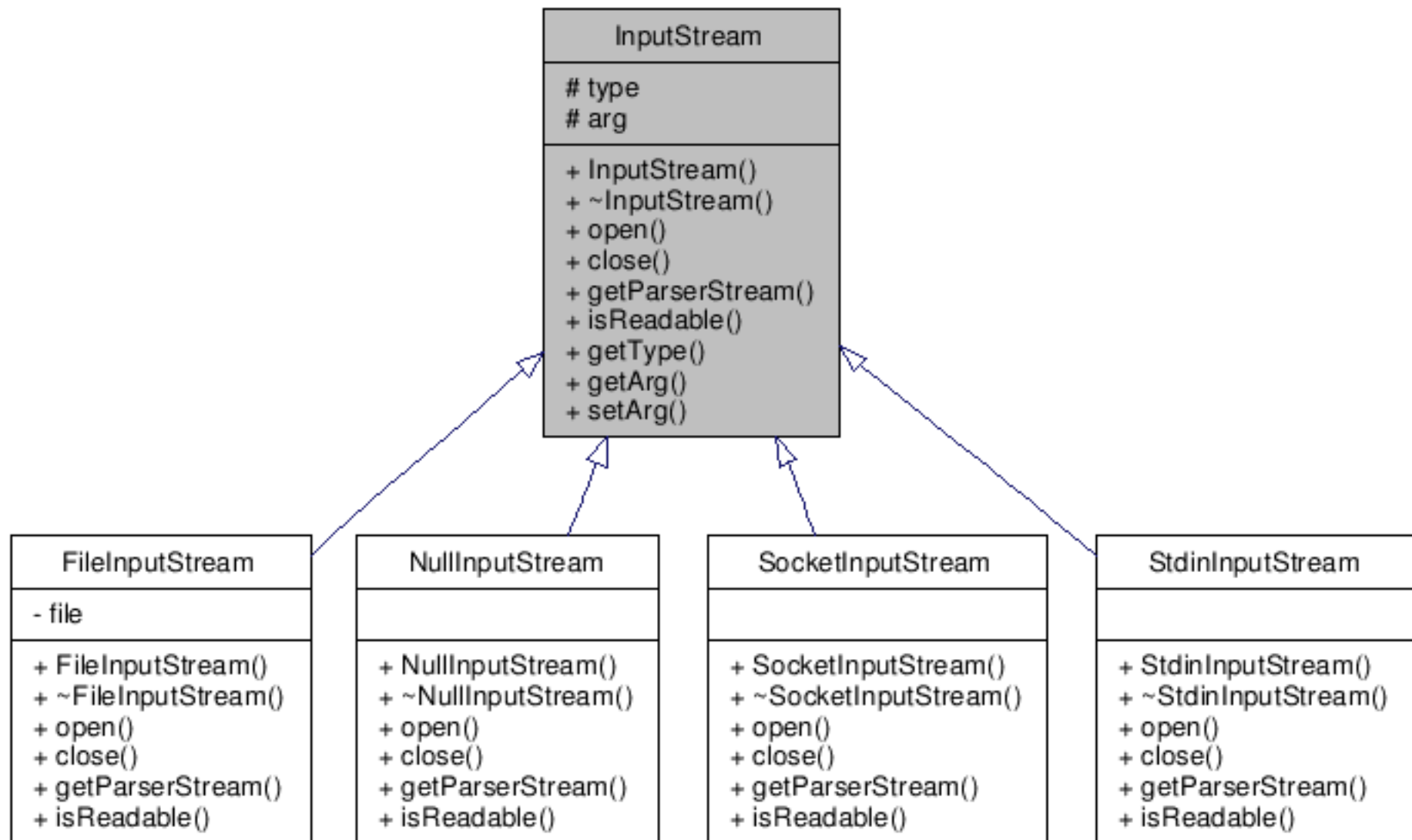
    System.out.println(novaPasta.getAbsolutePath());
}
```

SAIDA



Lendo conteúdo de arquivos

Lendo arquivo



Lendo arquivo

- A classe `InputStream`, é uma classe abstrata;
- Para lermos de um arquivo físico, deve-se utilizar a classe concreta `FileInputStream`;
- Quando trabalhamos com `java.io`, diversos métodos lançam ***IOException***, que é uma exception do tipo checked - o que nos obriga a tratá-la ou declará-la.

Lendo arquivo

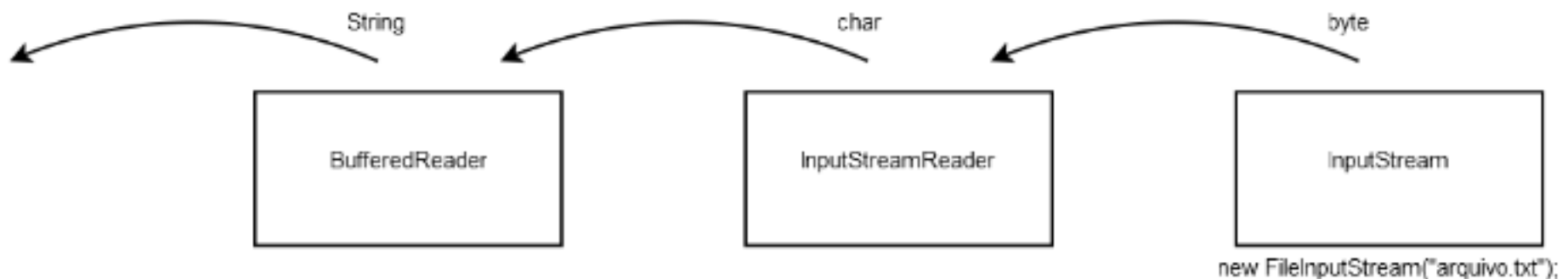
- Para recuperar um caractere, precisamos traduzir os bytes com o encoding dado para o respectivo código unicode, isso pode usar um ou mais bytes.
- Escrever esse decodificador é muito complicado, quem faz isso por você é a classe ***InputStreamReader***.

Lendo arquivo

- Apesar da classe abstrata `Reader` já ajudar no trabalho de manipulação de caracteres, ainda seria difícil pegar uma `String`.
- 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***:


Lendo arquivo

- Na pratica o ***InputStream*** lê o arquivo na forma de byte, o ***InputStreamReader*** converte de byte para char e por fim ***BufferedReader*** transforma o conteúdo para String:



Lendo arquivo

- Para exemplificar vamos ver o conteúdo do arquivo: arquivoEntrada.txt:

 arquivoEntrada.txt ✕

```
1 Este é um arquivo de teste para a leitura de arquivos.  
2 Esta é a segunda linha do arquivo.  
3 Esta é a terceira linha do arquivo.
```

Lendo arquivo

```
public void lerConteudoArquivo() {  
    InputStream leitorByte = null;  
    InputStreamReader leitorCaracter = null;  
    BufferedReader leitorPalavras = null;  
}
```

Declaração dos atributos de leitura

```
try {
```

```
    leitorByte = new FileInputStream("conteudo/arquivoEntrada.txt");  
    leitorCaracter = new InputStreamReader(leitorByte);  
    leitorPalavras = new BufferedReader(leitorCaracter);
```

```
    String s = leitorPalavras.readLine();
```

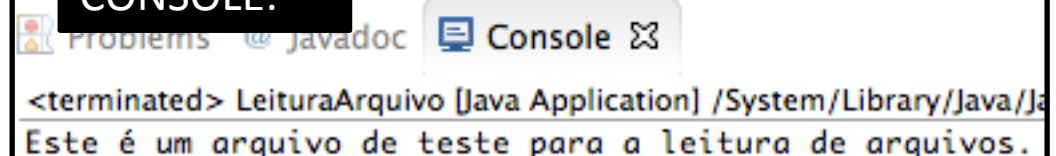
Lê o conteúdo e submete para uma String.

```
    System.out.println(s);  
} catch (FileNotFoundException e) {  
    System.err.println(e);  
} catch (IOException e) {  
    System.err.println(e);  
} finally {
```

```
    try {  
        if(leitorByte != null) {  
            leitorByte.close();  
        }  
        if(leitorCaracter != null) {  
            leitorCaracter.close();  
        }  
        if(leitorPalavras != null) {  
            leitorPalavras.close();  
        }  
    } catch (Exception e) {}  
}
```

Todo recurso aberto deve ser fechado após o uso

CONSOLE:



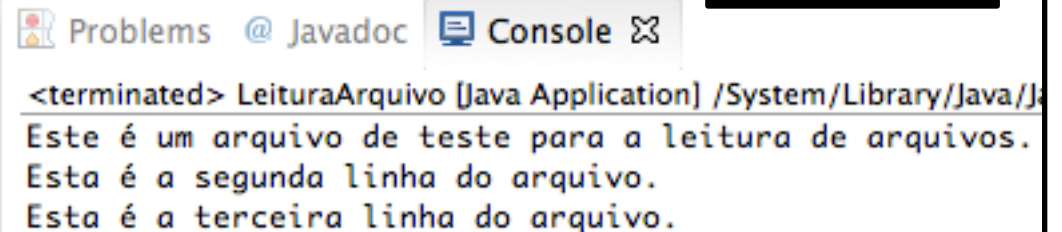
The screenshot shows an IDE window with tabs for 'Problems', 'Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a Java application. The text in the console is: '<terminated> LeituraArquivo [Java Application] /System/Library/Java/Ja' followed by a new line and 'Este é um arquivo de teste para a leitura de arquivos.'

Lendo arquivo

- Percebe-se que o método ***readLine()*** retorna o valor de uma linha do arquivo.
- Pode-se criar um laço para obter as demais linhas:

```
leitorByte = new FileInputStream("conteudo/arquivoEntrada.txt");  
leitorCaracter = new InputStreamReader(leitorByte);  
leitorPalavras = new BufferedReader(leitorCaracter);  
  
String linha = leitorPalavras.readLine();  
  
while(linha != null) {  
    System.out.println(linha);  
    linha = leitorPalavras.readLine();  
}
```

CONSOLE:



The screenshot shows an IDE window with tabs for 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a Java application named 'LeituraArquivo'. The output consists of three lines: a header line indicating the file path, followed by two lines of test data. The text is as follows:

```
<terminated> LeituraArquivo [Java Application] /System/Library/Java/J...  
Este é um arquivo de teste para a leitura de arquivos.  
Esta é a segunda linha do arquivo.  
Esta é a terceira linha do arquivo.
```

Escrevendo em arquivo

Escrevendo em arquivo

- Da mesma forma que para leitura temos 3 etapas, para a escrita não é diferente.
- Escrita dos bytes:
 - *OutputStream*
- Conversão para caracteres:
 - *OutputStreamWriter*
- Trata valor literal:
 - *BufferedWriter*

Escrevendo em arquivo

```
public void escreverConteudoArquivo() {
    OutputStream escritorByte = null;
    OutputStreamWriter escritorCaracter = null;
    BufferedWriter escritorPalavras = null;

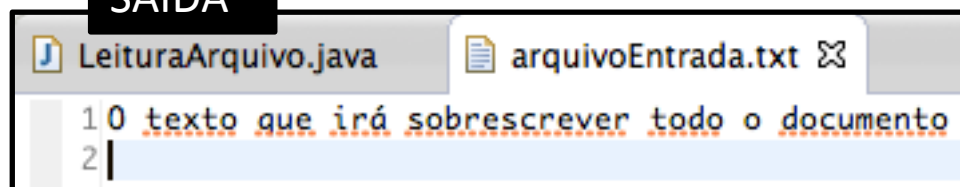
    try {
        escritorByte = new FileOutputStream("conteudo/arquivoEntrada.txt");
        escritorCaracter = new OutputStreamWriter(escritorByte);
        escritorPalavras = new BufferedWriter(escritorCaracter);

        String linha = "0 texto que irá sobrescrever todo o documento";

        //Escreve a linha no arquivo
        escritorPalavras.write(linha);
        //Adiciona uma nova linha
        escritorPalavras.newLine();
        //Atualizar os dados no arquivo
        escritorPalavras.flush();

    } catch (FileNotFoundException e) {
        System.err.println(e);
    } catch (IOException e) {
        System.err.println(e);
    } finally {
        try {
            if(escritorPalavras != null) {
                escritorPalavras.close();
            }
            if(escritorCaracter != null) {
                escritorCaracter.close();
            }
            if(escritorByte != null) {
                escritorByte.close();
            }
        } catch (Exception e){}
    }
}
```

SAÍDA



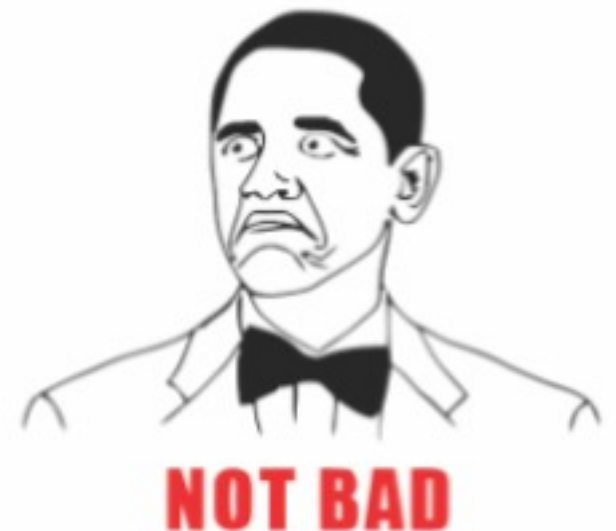
LeituraArquivo.java arquivoEntrada.txt

```
1 0 texto que irá sobrescrever todo o documento
2 |
```

Escrevendo em arquivo

É possível escrever apenas no final do arquivo,
sem sobrescrevê-lo?

REPOSTA: SIM

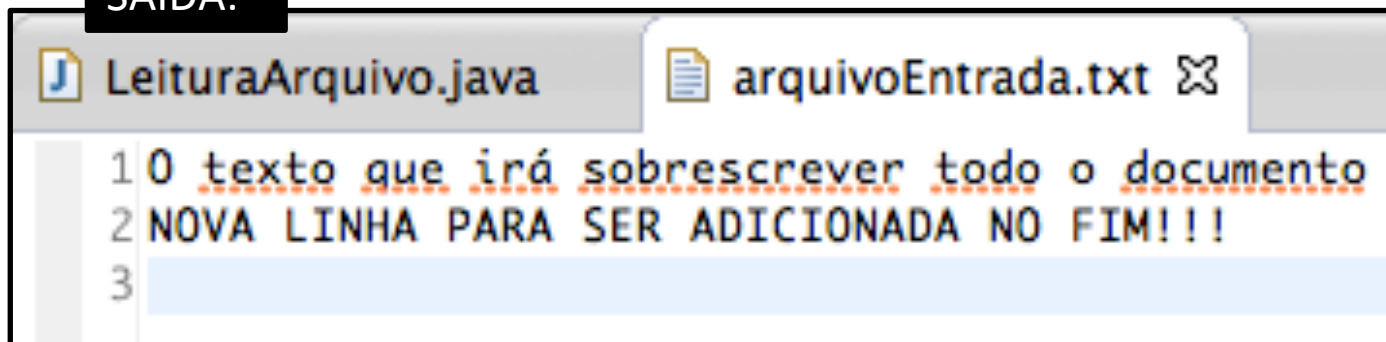


Escrevendo em arquivo

```
try {  
    escritorByte = new FileOutputStream("conteudo/arquivoEntrada.txt", true);  
    escritorCaracter = new OutputStreamWriter(escritorByte);  
    escritorPalavras = new BufferedWriter(escritorCaracter);  
  
    String linha = "NOVA LINHA PARA SER ADICIONADA NO FIM!!!";  
  
    //Escreve a linha no arquivo  
    escritorPalavras.write(linha);  
    //Adiciona uma nova linha  
    escritorPalavras.newLine();  
    //Atualizar os dados no arquivo  
    escritorPalavras.flush();  
}
```

Basta adicionar o parâmetro **true**, na construção do objeto da classe **FileOutputStream**.

SAIDA:



```
1 O texto que irá sobrescrever todo o documento  
2 NOVA LINHA PARA SER ADICIONADA NO FIM!!!  
3
```


Exercício 1

Lembra do exercício do PUB!?

Ele voltou!

Exercício 1



Um Pub solicitou o desenvolvimento de um sistema para auxiliar no sorteio de chopp para os clientes que estejam no estabelecimento.



Exercício 1

- Como funciona a promoção:
 - Todo cliente que ingressar no Pub, deve fornecer o seu número de celular (somente os números) para a atendente.
 - De tempos em tempos a atendente vai solicitar o sistema para que o mesmo sorteie um número de celular.
 - Após sorteado o número deve ser retirado da lista, garantindo que sempre serão sorteadas pessoas diferentes.

Exercício 1

- Requisitos do sistema:
 - Todo novo cliente que ingressar no Pub deve ser adicionado ao final da lista;
 - O sistema deve permitir a baixa do cliente quando o mesmo pagar a conta;
 - Deve-se localizar se o mesmo está na lista e remover o número do celular;
 - Caso o número do celular não exista na lista, deve-se informar que ele não foi encontrado.

Exercício 1

- Requisitos do sistema:
 - O sistema deve sortear os números disponíveis na lista, para isso deve-se utilizar a classe Random do Java a partir do índice da lista.
 - Ao sortear um número de celular o mesmo deve ser removido da lista;
 - Caso o sistema não possua nenhum número de celular cadastrado e seja solicitado um novo sorteio ou a baixa de um cliente, deve-se apresentar um mensagem informando que a lista está vazia.

Exercício 1

- Requisitos do sistema:
 - Deve-se escolher uma das Collections do Java para implementar estas operações;
 - **AS INFORMAÇÕES TAMBÉM DEVEM SER PERSISTIDAS EM ARQUIVOS.**
- Exemplo do uso da classe Random:

```
Random randomico = new Random();  
int aux = randomico.nextInt(6);  
  
System.out.println(aux);
```

Exercício 2

Cadastro de carros e motos

Exercício 2

- Foi solicitado a construção de um sistema para cadastro de carros e motos, onde esse sistema deve manter as informação em memória, mas também deve persisti-las em um arquivo texto.



Exercício 2

- Informações do Carro:
 - Marca;
 - Modelo;
 - Ano;
 - Cor;
 - Placa;
 - Numero de portas.

Exercício 2

- Informações da Moto:
 - Marca;
 - Modelo;
 - Ano;
 - Cor;
 - Placa.

Exercício 2

- O sistema de permitir as seguintes operações para motos e carros:
 - Inserir um novo veiculo;
 - Consultar se um veiculo existe;
 - Pela placa ou pelo modelo;
 - Remover um veículo;
 - Antes é necessário consulta e mostrar na tela;
 - Alterar os valores de uma veículo;
 - Antes é necessário consulta e mostrar na tela;

Exercício 2

- Não são permitidos veículos com a mesma placa (pode-se garantir isso pela estrutura de dados escolhida);
- Sobre o arquivo:
 - Cada linha pode ser um novo registro;
 - Pode-se montar uma estrutura baseada em | para dividir as informações de um registro;
 - Ex:
marca=fiat|modelo=uno|ano=2010|cor=preto|placa=asd1234|porta=4|