



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
ESTRUTURA DE DADOS ORIENTADA A OBJETOS
ADS1232
PROF. MSC. ANIBAL SANTOS JUKEMURA

PROGRAMAÇÃO ORIENTADA A OBJETOS [JAVA]

Agenda



- Arquivos

Arquivos: Comandos gerais

- A manipulação de arquivos em Java acontece de forma simples e rápida, pois a linguagem dispõe de classes que executam praticamente todas as operações necessárias para tanto:

`java.io.File`

- A classe File representa um arquivo ou diretório no sistema operacional. Importante saber que apenas REPRESENTA, não significa que o arquivo ou diretório realmente exista.

Arquivos: Comandos gerais

- Para instanciar um objeto do tipo File:

```
File arquivo = new File("/home/hallan/nome_do_arquivo.txt" );
```

```
File arquivo = new File("c:\\temp\\arquivo.txt" );
```

- Com o objeto instanciado, é possível fazer algumas verificações, como por exemplo se o arquivo ou diretório existe:

```
boolean existe = arquivo.exists();
```

Arquivos: Comandos gerais

- É possível criar um arquivo ou diretório:

```
//cria um arquivo vazio  
arquivo.createNewFile();
```

```
//cria um diretório  
arquivo.mkdir();
```

- É possível listar seus arquivos e diretórios através do método `listFiles()`, que retorna um vetor de `File`:

```
//Em caso de diretório, é possível listar seus arquivos e diretórios  
File [] arquivos = arquivo.listFiles();
```

Arquivos: Comandos gerais

- É possível também excluir o arquivo ou diretório através do método `delete()`. Uma observação importante é que, caso seja um diretório, para poder excluir, este tem de estar vazio:

```
//exclui o arquivo ou diretório  
arquivo.delete();
```

Arquivos: Comandos gerais

- As classes **FileWriter** e **BufferedWriter** servem para escrever em arquivos de texto.
- A classe **FileWriter** serve para escrever diretamente no arquivo, enquanto a classe **BufferedWriter**, além de ter um desempenho melhor, possui alguns métodos que são independentes de sistema operacional, como quebra de linhas.

Arquivos: Comandos gerais

- Para instanciar um objeto do tipo FileWriter:

```
//construtor que recebe o objeto do tipo arquivo  
FileWriter fw = new FileWriter( arquivo );
```

```
// construtor que recebe também como argumento se o  
// conteúdo será acrescentado ao invés de ser substituído
```

```
FileWriter fw = new FileWriter( arquivo, true );
```


Arquivos: Comandos gerais

- A criação do objeto `BufferedWriter`:

```
//construtor recebe como argumento o objeto do tipo  
FileWriter BufferedWriter bw = new BufferedWriter( fw );
```

Com o **bufferedwriter** criado, agora é possível escrever conteúdo no arquivo através do método **write()**:

```
//escreve o conteúdo no arquivo  
bw.write( "Texto a ser escrito no txt" );
```

```
//quebra de linha  
bw.newLine();
```

Arquivos: Comandos gerais

- Após escrever tudo que queria, é necessário **fechar os buffers** e informar ao sistema que o arquivo não está mais sendo utilizado:

```
//fecha os recursos  
bw.close();  
fw.close();
```

Arquivos: Comandos gerais

- **java.io.FileReader** e **java.io.BufferedReader**
- As classes **FileReader** e **BufferedReader** servem para ler arquivos em formato texto.
- A classe **FileReader** recebe como argumento o objeto **File** do arquivo a ser lido:

```
//construtor que recebe o objeto do tipo arquivo  
FileReader fr = new FileReader( arquivo );
```

Arquivos: Comandos gerais

- A classe **BufferedReader**, fornece o método **readLine()** para leitura do arquivo:

```
//construtor que recebe o objeto do tipo FileReader  
BufferedReader br = new BufferedReader( fr );
```

Arquivos: Comandos gerais

- Para ler o arquivo, basta utilizar o método **ready()**, que retorna se o arquivo tem mais linhas a ser lido, e o método **readLine()**, que retorna a linha atual e passa o buffer para a próxima linha:

```
//enquanto houver mais linhas
while( br.ready() ){
    //lê a próxima
    linha String linha = br.readLine();
    //faz algo com a linha
}
```

Arquivos: Comandos gerais

- Após ler tudo que queria, é necessário **fechar os buffers** e informar ao sistema que o arquivo não está mais sendo utilizado:

```
//fecha os recursos  
br.close();  
fr.close();
```

Arquivos



```
1  import java.io.*;
2
3  public class TesteArq {
4      public static void main(String[] args) {
5          File arquivo = new File("C:\\Users\\lucil\\Desktop\\codigos\\Java\\ED\\Arquivos\\frutas.txt");
6
7          if (!arquivo.exists()) {
8              try {
9                  arquivo.createNewFile();
10             } catch (IOException e) {
11                 throw new RuntimeException(e);
12             }
13         } else
14             System.out.println("Arquivo existente.");
15
16
17         try {
18             FileWriter fw = new FileWriter( arquivo, true );
19             BufferedWriter bw = new BufferedWriter( fw );
20
21             bw.write( "Banana" );
22             //quebra de linha
23             bw.newLine();
24             bw.write( "Morango" );
25             //quebra de linha
26             bw.newLine();
27             bw.write( "Uva" );
28             bw.close();
29             fw.close();
30         } catch (IOException e) {
31             throw new RuntimeException(e);
32         }
33     }
34 }
```



```
33
34
35     FileReader fr = new FileReader( arquivo );
36     BufferedReader br = new BufferedReader( fr );
37     System.out.print("Frutas: [ ");
38     while( br.ready() ){
39         String linha = br.readLine();
40         System.out.print(linha + " ");
41     }
42     System.out.println("]");
43     br.close();
44     fr.close();
45 } catch (FileNotFoundException e) {
46     throw new RuntimeException(e);
47 } catch (IOException e) {
48     throw new RuntimeException(e);
49 }
50
51
52 }
53 }
```



```
Frutas: [ Banana Morango Uva ]
```

```
Arquivo existente.
Frutas: [ Banana Morango UvaBanana Morango Uva ]
```