



#R4E

Software Developer

# Algoritmia e Programação

Ficheiros



# Conteúdo

- Ler Ficheiros de Texto
- Escrever Ficheiros de Texto
- Casos de Uso

# Ficheiros

**“Conjunto de dados gravados no suporte físico de um sistema informático”**

- É um conjunto de informação guardado num suporte de armazenamento (secundário) de forma durável.
- “Pode ser considerado como um objeto, possuindo um nome que o identifica, atributos e valores”.
- Pode conter dados estruturados ou não:
  - Não estruturados - possuem apenas sequências de bytes.
  - Estruturados - podem ser organizados em registos ou em árvore.
- Um ficheiro de texto é composto por caracteres

# Ficheiros de Texto - Java

## File

- É uma representação abstrata de nomes de caminho de ficheiros e diretórios no disco.

- **Métodos:**

- public boolean exists()
- public boolean isFile()
- public boolean isDirectory()
- public String getPath()
- public String getName()
- ...

Separadores:

/ (UNIX + Windows)

\\ (Windows)



**File** file = new **File**(" /temp/samplefile1.txt");

**File** file = new **File**("c:\\temp\\samplefile1.txt");

# Ficheiros de Texto - Escrita e Leitura

Existem várias maneiras para escrever e ler de ficheiros de texto.

- **Escrever**

- FileWriter
- PrintWriter
- BufferedWriter
- Formatter

- **Ler**

- FileReader
- BufferedReader
- Scanner

# Ficheiros de Texto - Escrita - PrintWriter

(java.util.PrintWriter)

- Permite escrever texto formatado num ficheiro de texto.
- Permite usar métodos de formatação tal como em System.out: print(), println(), printf()

# Ficheiros de Texto - Escrita - PrintWriter

```
public static void usingPrintWriter() throws FileNotFoundException {
```

Exceção Verificável

```
String fileContent = "Hello! Adoro Programar em Java";
```

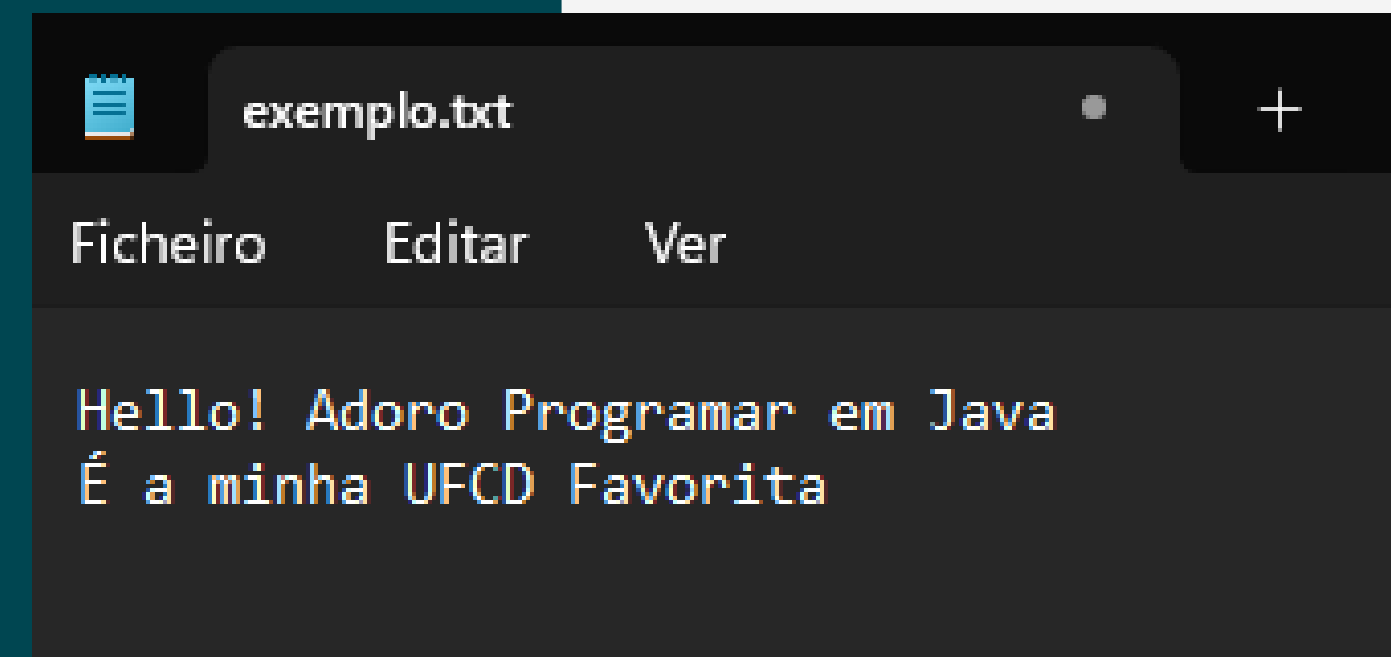
```
File file = new File("C:\\temp\\exemplo.txt");  
PrintWriter printWriter = new PrintWriter(file);
```

Criar Ficheiro

```
printWriter.println(fileContent);  
printWriter.println("É a minha UFCD Favorita");
```

```
printWriter.close();  
}
```

Fechar Ficheiro



# Ficheiros de Texto - Escrita - Formatter

(java.util.Formatter)

- Permite escrever texto formatado num ficheiro de texto.
- Permite usar o método de formatação `format()`.



# Ficheiros de Texto - Escrita - Formatter

```
public static void usingFormatter() throws FileNotFoundException {
```

Exceção Verificável

```
String fileContent = "Hello! Adoro Programar em Java";
```

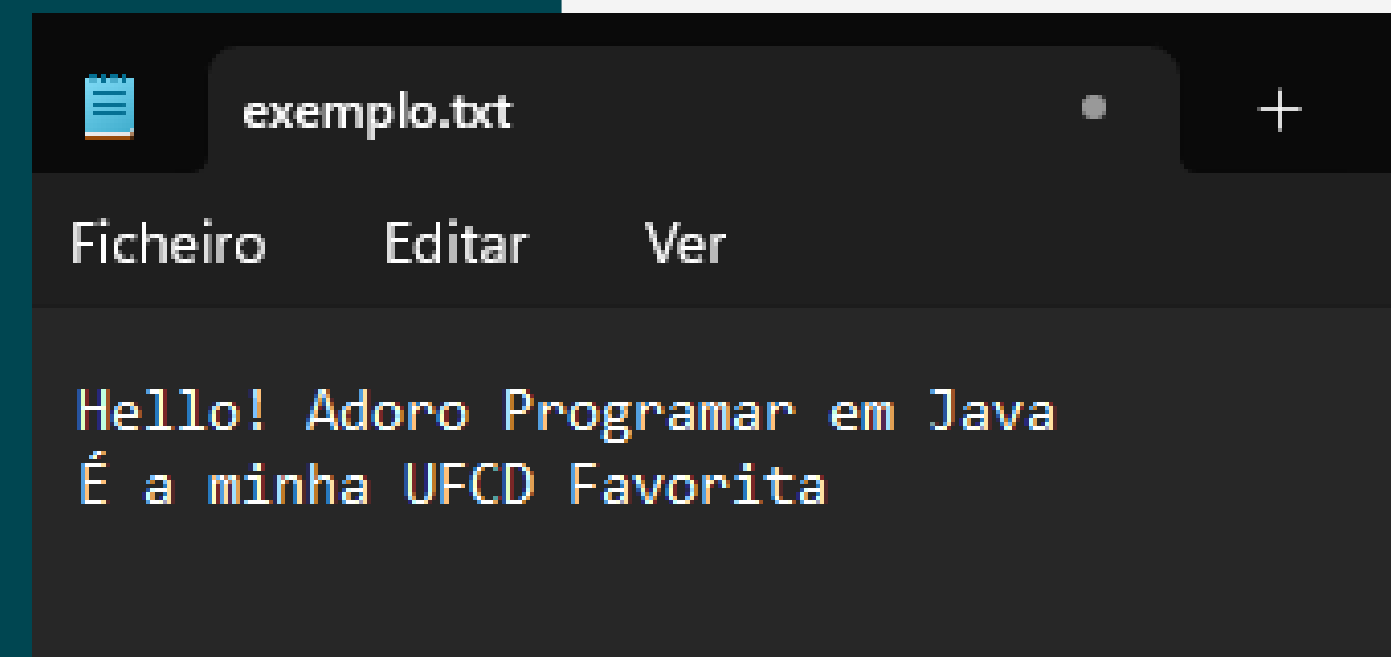
```
File file = new File("C:\\temp\\exemplo.txt");  
Formatter formatter = new Formatter(file);
```

Criar Ficheiro

```
formatter.format(fileContent + "\n");  
formatter.format("É a minha UFCD Favorita");
```

```
formatter.close();  
}
```

Fechar Ficheiro



# Ficheiros de Texto - Leitura - Scanner

(java.util.Scanner)

- Permite converter texto para tipos primitivos e strings (exceto char).
- O texto pode ser obtido de diversas fontes, por exemplo:
  - Teclado (System.in)
  - Strings
  - **Ficheiros**
- Permite separar o texto em tokens, que são sequências de caracteres separados por delimitadores.
- Por omissão, os delimitadores são: espaço, tab e mudança de linha (enter).
- Os tokens resultantes podem ser convertidos em valores de diferentes tipos usando os vários métodos “next...”: nextInt(), nextDouble(), next(), nextLine()...

# Ficheiros de Texto - Leitura - Scanner

```
public static void usingScanner() throws FileNotFoundException {
```

Exceção Verificável

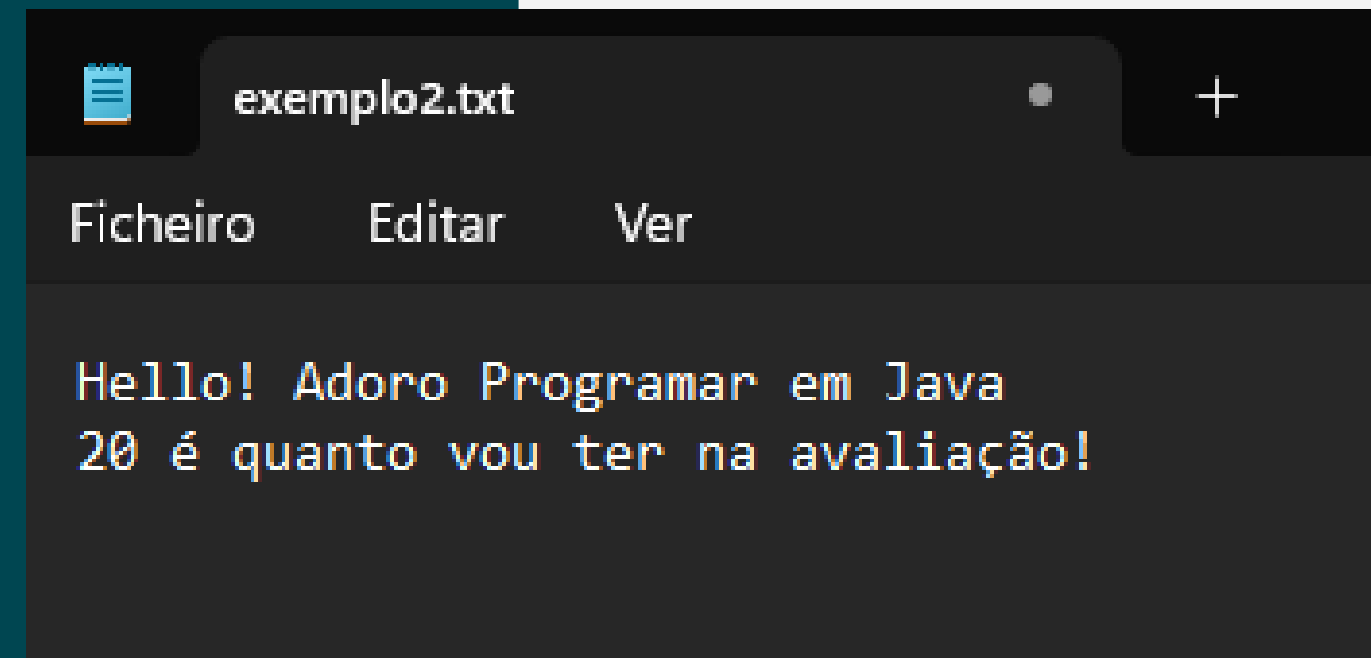
```
File file = new File("C:\\temp\\exemplo2.txt");  
Scanner sc = new Scanner(file);
```

Abrir Ficheiro

```
System.out.println( sc.nextLine() );  
System.out.println( sc.nextInt() );  
System.out.println( sc.next() );  
System.out.println( sc.nextLine() );
```

```
sc.close( );  
}
```

Fechar Ficheiro



CONSOLA IDE

```
> Hello! Adoro Programar em Java  
> 20  
> é  
> quanto vou ter na avaliação!
```

# Ficheiros de Texto - Verificar Existência de Informação - Scanner

(java.util.Scanner)

- É possível verificar se o ficheiro contém mais informação através de métodos booleanos.
- Métodos de Verificação:
  - hasNext( )
  - hasNextInt( )
  - hasNextDouble( )
  - hasNextFloat( )
  - hasNextLine( )
  - ...

# Ficheiros de Texto - Nota Importante

- Se tentar aceder a um ficheiro de texto que não existe, é lançada a exceção **FileNotFoundException**
- Deve sempre fechar um ficheiro após a utilização ( `close()` ) para libertar recursos.
- O método `close()` invoca o método `flush()` para garantir que antes de libertar os recursos, sejam primeiro gravados no ficheiro quaisquer bytes em buffer.

Ler e visualizar a informação de todos os items comprados.

# Ficheiros de Texto - Problema 1

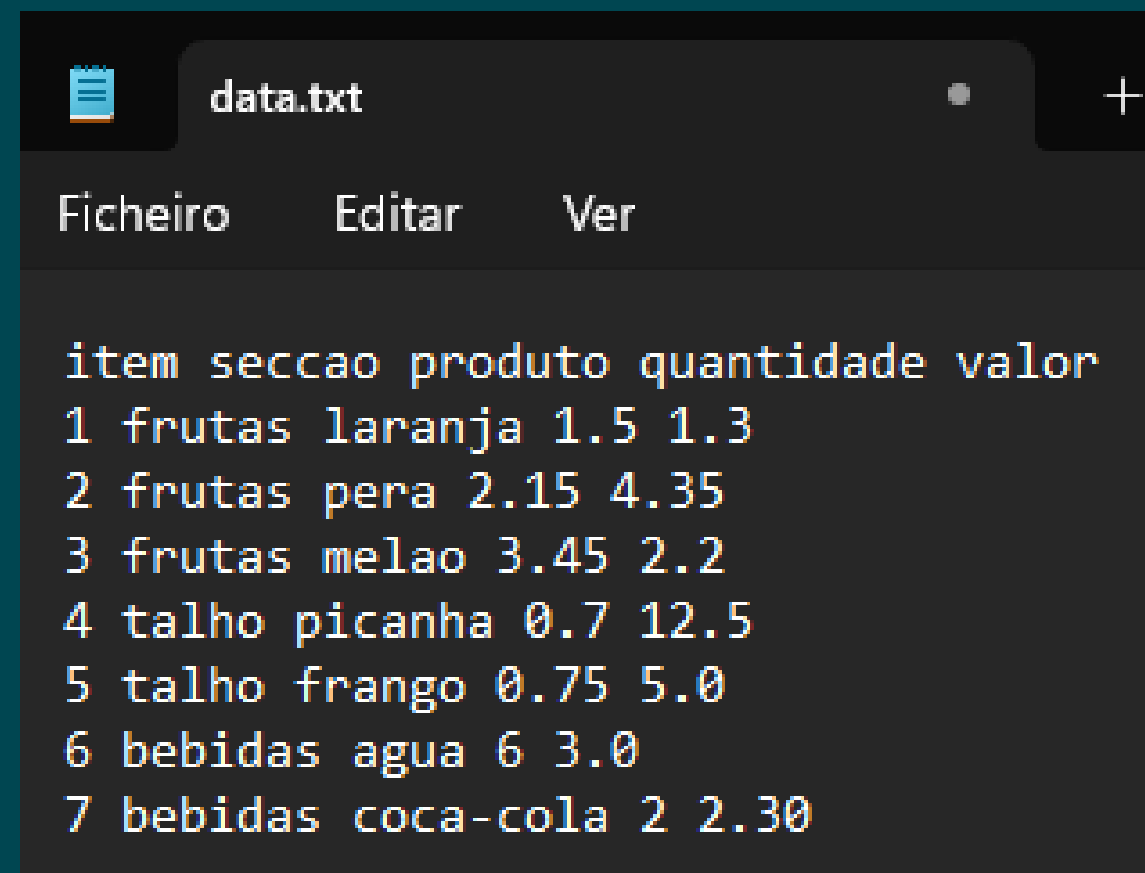
```
public static void lerLinhasInteirasdoFicheiro() throws FileNotFoundException {
```

```
    Scanner in = new Scanner( new File("data.txt") );
```

```
    String linha = in.nextLine();    //linha do cabecalho
```

```
    while (in.hasNextLine()) {  
        linha = in.nextLine();  
        System.out.println(linha);  
    }
```

```
    in.close( );  
}
```



The screenshot shows a text editor window with a dark theme. The title bar says 'data.txt'. Below the title bar is a menu bar with 'Ficheiro', 'Editar', and 'Ver'. The main text area contains the following content:

```
item seccao produto quantidade valor  
1 frutas laranja 1.5 1.3  
2 frutas pera 2.15 4.35  
3 frutas melao 3.45 2.2  
4 talho picanha 0.7 12.5  
5 talho frango 0.75 5.0  
6 bebidas agua 6 3.0  
7 bebidas coca-cola 2 2.30
```

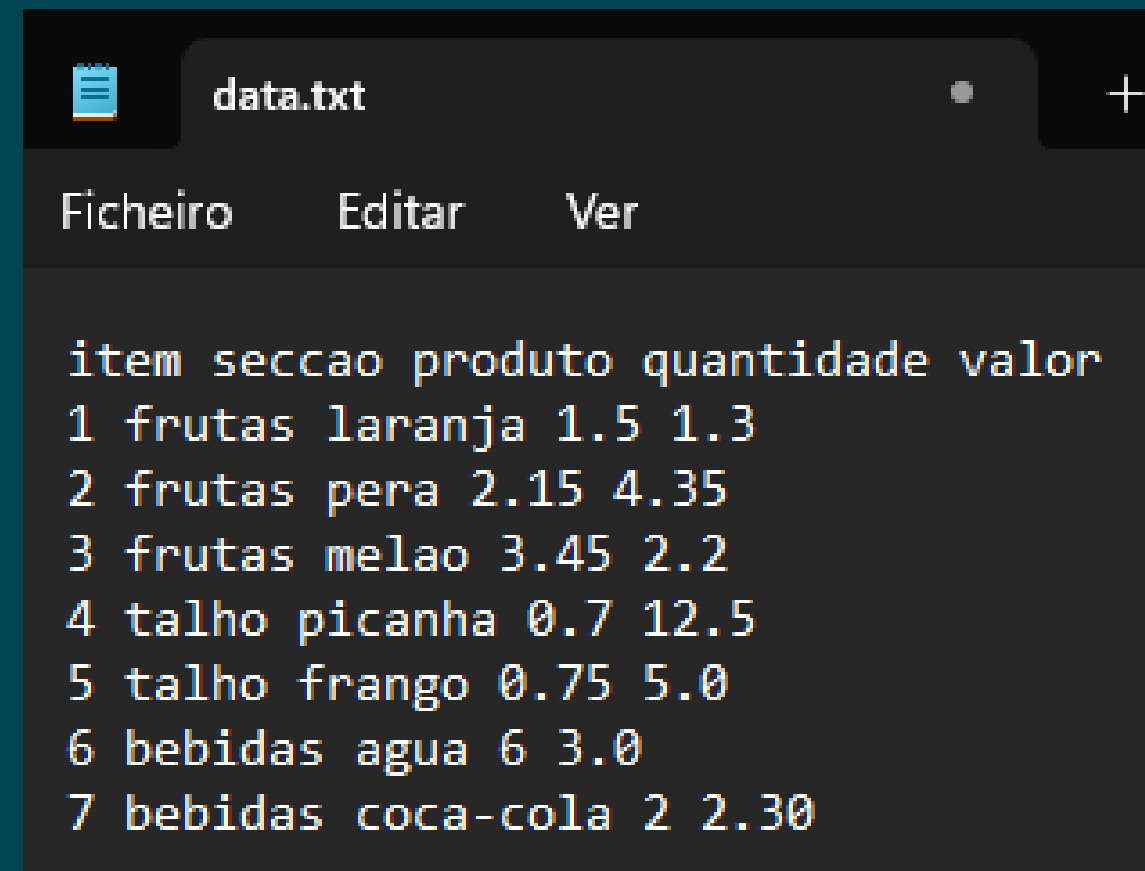
## CONSOLA IDE

```
> 1 frutas laranja 1.5 1.3  
> 2 frutas pera 2.15 4.35  
> 3 frutas melao 3.45 2.2  
> 4 talho picanha 0.7 12.5  
> 5 talho frango 0.75 5.0  
> 6 bebidas agua 6 3.0  
> 7 bebidas coca-cola 2 2.30
```

Visualizar o valor total gasto nas compras.

# Ficheiros de Texto - Problema 2

```
public static void calcularTotalDasCompras() throws FileNotFoundException {  
  
    Scanner in = new Scanner( new File("data.txt") );  
  
    String linha = in.nextLine(); //linha do cabecalho  
  
    int item;  
    String seccao, produto;  
    double quantidade, valor, total = 0;  
  
    while (in.hasNextLine()) {  
        item = in.nextInt();  
        seccao = in.next();  
        produto = in.next();  
        quantidade = in.nextDouble();  
        valor = in.nextDouble();  
  
        total += valor;  
    }  
  
    System.out.println("Total: " + total);  
    in.close( );  
}
```



The screenshot shows a text editor window titled 'data.txt'. The menu bar includes 'Ficheiro', 'Editar', and 'Ver'. The text content is as follows:

item	seccao	produto	quantidade	valor
1	frutas	laranja	1.5	1.3
2	frutas	pera	2.15	4.35
3	frutas	melao	3.45	2.2
4	talho	picanha	0.7	12.5
5	talho	frango	0.75	5.0
6	bebidas	agua	6	3.0
7	bebidas	coca-cola	2	2.30

CONSOLA IDE

Total: 30.65000000000000

# String.split( )

O método split() corta uma string sempre que encontra nela uma expressão idêntica à expressão do parâmetro do método split e constrói um array resultado com as partes cortadas da string.

String linha = "1|frutas|laranja|1.5|1.3"

String[ ] itensDaLinha = linha.split(" ");

Corta a String sempre que encontra um espaço

Resulta num array de Strings

itensDaLinha	
"1"	0
"frutas"	1
"laranja"	2
"1.5"	3
"1.3"	4



# String.split( ) - Exemplo

Imprimir todas as palavras de uma String (que estão separadas por espaços).

```
String linha = "1 frutas laranja 1.5 1.3"
```

**Delimitador**

```
String[ ] itensDaLinha = linha.split(" ");
```

```
for(int i=0; i < itensDaLinha.length; i++){  
    System.out.println( itensDaLinha[i] );  
}
```

CONSOLA IDE

```
> 1  
> frutas  
> laranja  
> 1.5  
> 1.3
```

**itensDaLinha**

"1"	0
"frutas"	1
"laranja"	2
"1.5"	3
"1.3"	4

Visualizar o valor total gasto numa secção específica

# Ficheiros de Texto - Problema 3

```
public static void totalGastoSeccao(String seccao) throws FileNotFoundException {
```

```
    Scanner in = new Scanner( new File("data.txt") );
```

```
    int item;
```

```
    String seccao, produto;
```

```
    double total = 0;
```

```
    while (in.hasNextLine()) {
```

```
        linha = in.nextLine();
```

```
        String[] itensDaLinha = linha.split(" ");
```

```
        if (seccao.equals(itensDaLinha[1])) {
```

```
            total += Double.parseDouble(itensDaLinha[4]);
```

```
        }
```

```
    }
```

```
    System.out.println("Seccao [" + seccao + "] total= " + total);
```

```
    in.close( );
```

```
}
```



data.txt



Ficheiro

Editar

Ver

```
item seccao produto quantidade valor
1 frutas laranja 1.5 1.3
2 frutas pera 2.15 4.35
3 frutas melao 3.45 2.2
4 talho picanha 0.7 12.5
5 talho frango 0.75 5.0
6 bebidas agua 6 3.0
7 bebidas coca-cola 2 2.30
```

CONSOLA IDE

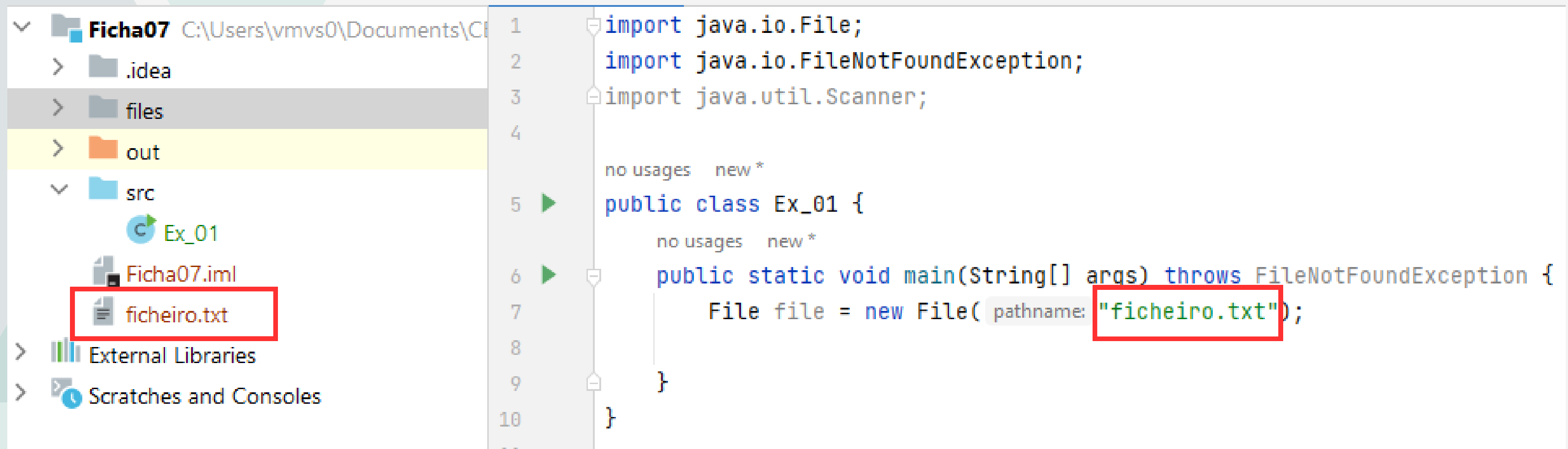
Seccao [frutas] total = 7.95

# Ficheiros de Texto - Como abrir no IDE IntelliJ

- No IDE IntelliJ é possível abrir ficheiros, num programa, sem explicitar o caminho absoluto.
- Por ausência de caminho absoluto, o nosso root (raíz), será a pasta do projeto.
- Não esquecer de colocar a extensão (por exemplo: .txt .csv , etc).

# Ficheiros de Texto - Como abrir no IDE IntelliJ

- Exemplo de como abrir um ficheiro que se encontra diretamente na pasta do projeto. (Não colocar ficheiros na pasta src)
- Note que a ausência de caminho absoluto implica que nos encontramos na pasta do projeto.



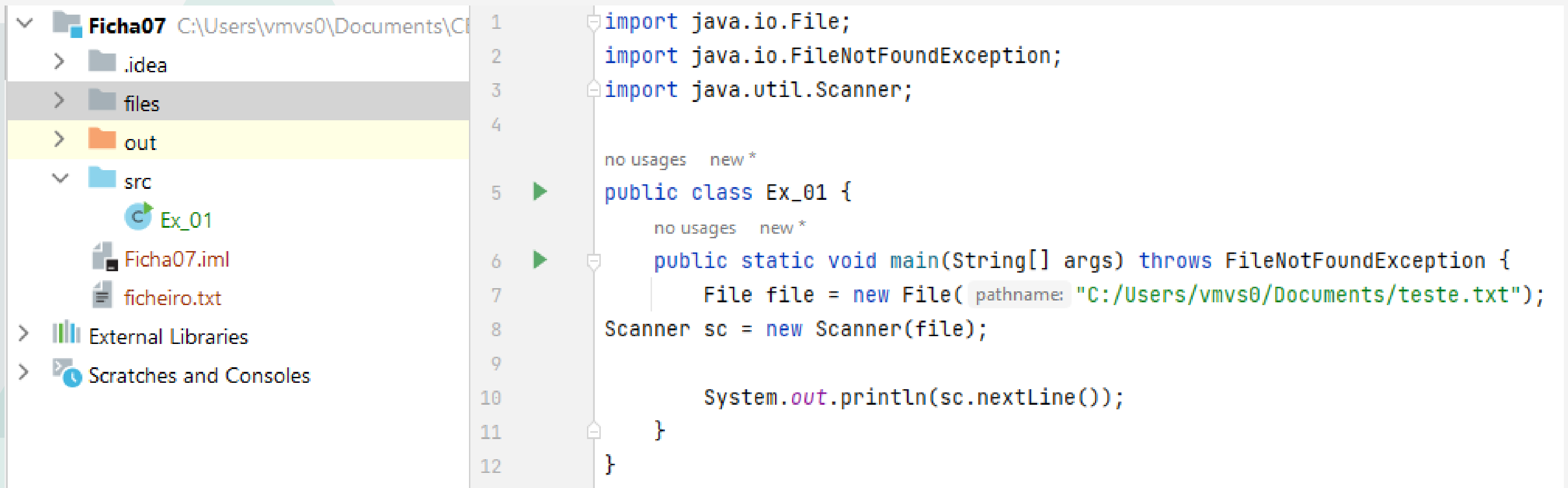
# Ficheiros de Texto - Como abrir no IDE IntelliJ

- Exemplo de como abrir um ficheiro que se encontra numa pasta dentro da pasta do projeto. (Não colocar pastas dentro da pasta src)
- Note que a ausência de caminho absoluto implica que nos encontramos na pasta do projeto.



# Ficheiros de Texto - Como abrir no IDE IntelliJ

- Exemplo de como abrir um ficheiro se encontra noutra pasta do computador.
- Note que o caminho absoluto tem de estar explícito.





#R4E

Software Developer

# Algoritmia e Programação

Ficheiros

