

Projeto 1

File Tools



Isabel Nunes

Unidade Curricular de
Laboratório de Programação

2020/2021

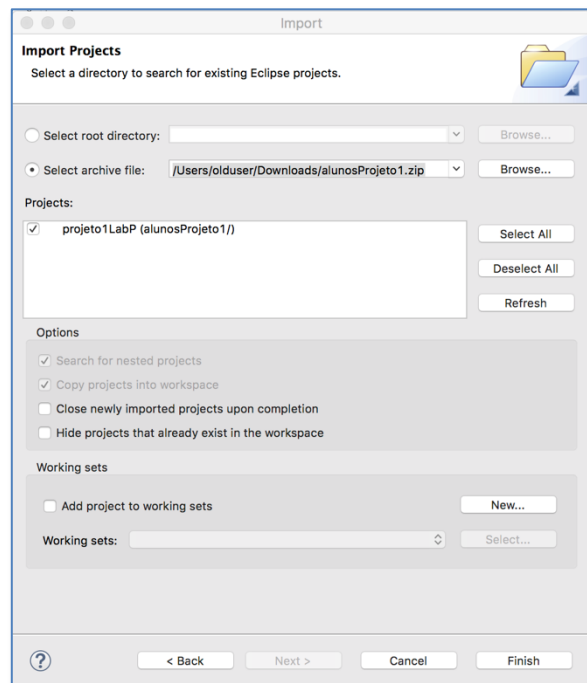
Objetivos

- Leitura e manipulação de textos contidos em ficheiros;
- Manipulação de *strings*.

Antes de Começar

Importar o projeto

- Fazer *download* do ficheiro `alunosProjeto1.zip` na página de LabP;
- No Eclipse,
 - escolher **File** → **Import** → **Existing Projects in Workspace**
 - escolher a opção **Select archive file** e clicar em “Browse”
 - encontrar e seleccionar o `alunosProjeto1.zip` e carregar em **Open**
 - depois de carregar em “Finish”, verá aparecer o projeto `projeto1LabP` na secção **Package Explorer** do Eclipse (tem erros pois falta a classe que os alunos vão ter que desenvolver)



Como poderá verificar, depois destes passos, foi criado um projeto `projeto1LabP` que contém:

- Na pasta `src`, a classe `RunProject1`;
- Na pasta `testes`, um ficheiro que permite executar todos os testes (`TestsProject1`);
- Tem ainda sete ficheiros de texto na raiz do projeto, que irão ser utilizados nas invocações dos métodos feitas no `main` da classe `RunProject1`;
- Contém também as bibliotecas do Java e do JUnit.

Para aferir (em parte) a correção do seu projeto deve usar a classe `RunProject1.java`, bem como a classe de testes JUnit `TestsProject1.java`.

Pode ainda fazer *download* na página de LabP do ficheiro `FicheirosOutput.zip` que contém os ficheiros de texto que os métodos pretendidos devem produzir quando o `main` da classe `RunProject1` é executado. Pode comparar estes ficheiros com os que obtém com a sua solução.

Muito Importante!

Quando temos um ciclo infinito que escreve para um ficheiro, rapidamente o disco fica cheio e a seguir não conseguimos escrever mais nada.

Isto aconteceu a alguns alunos na primeira semana e nem sequer conseguiam construir o zip para submissão.

Para evitar esta situação:

- Enquanto estão a desenvolver o método escrevem para o `System.out`;
- Se houver um ciclo infinito, rapidamente se aperceberão disso e poderão terminar a execução (basta clicar no quadrado encarnado na janela da consola);
- Quando tiverem a certeza que o método já está correto, alteram a escrita para ficheiro, usando então um objeto `PrintWriter`.

Algumas informações úteis

De modo a poder realizar este projeto deverá recordar como utilizar as classes que lhe permitem ler e escrever dados em ficheiros de texto, nomeadamente `Scanner` e `PrintWriter`, e também as classes `String` e `StringBuilder`.

Pode consultar os dois apontamentos (sobre o `Scanner` e sobre `Strings` e `StringBuilder`) acessíveis na página de LabP.

Neste projeto vai ter que ler as linhas de um ficheiro de texto e escrever parte dessa informação, eventualmente alterada, noutra ficheiro de texto.

O excerto de programa seguinte é um exemplo simples e incompleto de leitura de um ficheiro linha a linha.

```
public class LeituraLinhasFicheiro {

    public static void main(String args[]) {

        Scanner in = new Scanner (new File("meuFich.txt"));
        PrintWriter out = new PrintWriter("teuFich.txt");

        while (in.hasNextLine()) {
            String line = in.nextLine();

            // usar a string line para obter a informação
            // necessaria para escrever no ficheiro
            ...

            // Se for newLine essa nova informação:
            out.println(newLine);
        }

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

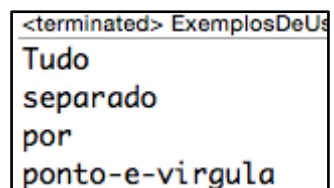
Para cada método que lhe é pedido mais adiante neste enunciado, tente perceber:

- se pode trabalhar cada linha do ficheiro como uma única *string* (por exemplo, se tiver que criar uma cópia de algumas ou de todas as linhas do ficheiro de entrada, linha a linha),
- se tem que trabalhar cada linha do ficheiro palavra a palavra (consegue aceder às várias palavras usando, por exemplo, o método `split` da classe `String` com os separadores adequados ao caso específico – ver exemplo já a seguir) ou
- se tem que trabalhar cada linha do ficheiro carácter a carácter, usando para isso, por exemplo, o método `charAt` já seu conhecido.

Use objetos do tipo `StringBuilder` sempre que precisar de construir uma *string* passo a passo.

Relembrando: Em algumas classes, como por exemplo na classe `String`, existem métodos (por exemplo `split`) que permitem que, dado um **padrao** (isto é, uma *string* que representa uma expressão regular) como argumento, se possa dividir a `String` em vários pedaços de acordo com o **padrao** dado.

```
public class ExemplosDeUso {  
  
    public static void main(String args[]) {  
        String linha = "Tudo;separado;por;ponto-e-virgula";  
  
        String[] resultados = linha.split(";");  
  
        for (String str: resultados) {  
            System.out.println(str);  
        }  
    }  
}
```



```
<terminated> ExemplosDeUso  
Tudo  
separado  
por  
ponto-e-virgula
```

O argumento que o método `split` recebe representa um padrão que é usado como separador dos elementos da `String`. Se as palavras de interesse também pudessem estar separadas por espaços (um ou mais), pontos e vírgulas, por exemplo, deveríamos ter usado:

```
linha.split("\\s*[ ,.;]\\s*")
```

Na verdade usámos uma expressão regular que pode ser muito simples ou muito complexa. Mais à frente no semestre haverá uma aula dedicada ao assunto.

Enunciado

A equipa TUTEDECC ("Tem Um TEXto Demasiado Claro? Contacte-nos!") é uma empresa que se dedica ao estudo de várias técnicas de tratamento de textos, quer para obscurecimento, quer para análise no contexto de aplicações criptográficas. Com este projeto pretende-se enriquecer o leque de técnicas à disposição da equipe da TUTEDECC criando novas formas de manipular texto.

ATENÇÃO 1: se, quando abrir os ficheiros de texto fornecidos, encontrar caracteres estranhos, faça o seguinte: com o ficheiro aberto e selecionado, escolher Menu **Edit** → **Set Encoding** → **Other**, e seleccionar a opção UTF-8.

ATENÇÃO 2: não se esqueça que tem que fazer Refresh na janela Package Explorer para conseguir visualizar os ficheiros que o seu programa cria no ambiente do Eclipse. (com o projeto selecionado, botão direito do rato, escolher Refresh).

O que fazer então?

Deve desenvolver a classe `TUTEDECC` com os métodos descritos de seguida. Note que os parâmetros do tipo `String` identificam os **nomes** dos ficheiros.

- `static void copyPositionMultiple (String fileIn, int n, String fileOut) throws FileNotFoundException` que copia para o ficheiro de nome `fileOut` as linhas do ficheiro de texto de nome `fileIn` que estão numa posição que é múltipla de `n`, precedidas do número da linha, seguido de “.” e de espaço.

```
copyPositionMultiple(fileIn, n, fileOut)
```

se o ficheiro de nome `fileIn` contiver o texto:
as armas e os barões assinalados
que da ocidental praia lusitana
por mares nunca de antes navegados
passaram ainda além da taprobana
em perigos e guerras esforçados

e o valor do parâmetro `n` for 2

o ficheiro de nome `fileOut` deve conter:
2: que da ocidental praia lusitana
4: passaram ainda além da taprobana

- `static void vowelSubstitution (String fileIn, char c, String fileOut) throws FileNotFoundException` que copia as linhas do ficheiro de nome `fileIn` para o ficheiro de nome `fileOut` substituindo todas as vogais do texto pelo carácter `c`. Assuma que o ficheiro não tem vogais acentuadas.

```
vowelSubstitution(fileIn, c, fileOut)
```

se o ficheiro de nome `fileIn` contiver o texto:
as armas e os barões assinalados
que da ocidental praia lusitana
por mares nunca de antes navegados

e o valor do parâmetro `c` for `'o'`

o ficheiro de nome `fileOut` deve conter:
os ormos e os barões ossinalados
quo do ocodontol prooo losotono
por moros nonco do ontos novogodos

- `static int[] averageWordSize (String fileIn, int n) throws FileNotFoundException` que, assumindo que o ficheiro de texto com o nome `fileIn` tem pelo menos `n` linhas, lê as linhas do ficheiro e devolve um *array* com `n` elementos, contendo na posição `k` a média (arredondada às unidades) dos tamanhos das palavras da linha `k`. Assuma que os separadores de palavras podem ser apenas espaços, pontos ou vírgulas.

```
averageWordSize(fileIn, 3)
```

se o ficheiro de nome `fileIn` contiver o texto:
Agora vamos fazer
um teste. Ui ui.
algarismos significativos nestes tipos, pequeníssimos erros

o resultado do método deve ser:
{5, 3, 9}

- `static void oneWordPerLine (String fileIn, String fileOut)` `throws FileNotFoundException` que, para cada linha do ficheiro de nome `fileIn`, escreve uma linha no ficheiro de nome `fileOut` com o nº da linha e a última palavra dessa linha (se uma linha do ficheiro de entrada estiver vazia, a linha correspondente a escrever, terá apenas o nº da linha). Assuma que os separadores de palavras podem ser apenas espaços, pontos ou vírgulas.

```
oneWordPerLine(fileIn, fileOut)
```

se o ficheiro de nome `fileIn` contiver o texto:
As saudades que eu já tinha
Da minha alegre casinha
Tão modesta quanto eu
Meu Deus como é bom morar
Modesto primeiro andar
A contar vindo do céu

o ficheiro de nome `fileOut` deve conter:
1 tinha
2 casinha
3 eu
4 morar
5 andar
6 céu

- `static void rotateEveryWord (String fileIn, int n, String fileOut)` throws `FileNotFoundException` que escreve no ficheiro de nome `fileOut` linhas iguais às do ficheiro de nome `fileIn` mas em que todas as palavras sofreram a seguinte transformação: os `n` últimos caracteres da palavra original passaram para as `n` primeiras posições da palavra já transformada. As palavras cujo tamanho seja menor ou igual a `n`, ficam como estavam. Assuma que o texto no ficheiro de nome `fileIn` apenas tem letras e espaços.

```
rotateEveryWord(fileIn, n, fileOut)
```

se o ficheiro de nome `fileIn` contiver o texto:

```
as armas e os barões assinalados
que da ocidental praia lusitana
por mares nunca de antes navegados
```

e o valor do parâmetro `n` for 3

o ficheiro de nome `fileOut` deve conter:

```
as masar e os oesbar dosassinala
que da talociden aiapr analusit
por resma ncanu de tesan dosnavega
```

- `static void switchNextWordNextLine (String fileIn, String fileOut)` throws `FileNotFoundException` que copia linhas do ficheiro de nome `fileIn` para o ficheiro de nome `fileOut`, trocando, em cada par de linhas seguidas, a palavra na posição `k` da 1ª linha com a palavra na posição `k+1` da 2ª linha, para valores ímpares de `k`.

```
switchNextWordNextLine(fileIn, fileOut)
```

se o ficheiro de nome `fileIn` contiver o texto:

```
A1 A2 A3 A4 A5 A6 A7 A8
B1 B2 B3 B4 B5 B6 B7 B8
C1 C2 C3
D1 D2 D3 D4 D5 D6
E1 E2 E3 E4 E5 E6
F1 F2 F3
G1 G2 G3 G4
```

o ficheiro de nome `fileOut` deve conter:

```
B2 A2 B4 A4 B6 A6 B8 A8
B1 A1 B3 A3 B5 A5 B7 A7
D2 C2 D4
D1 C1 D3 C3 D5 D6
F2 E2 E3 E4 E5 E6
F1 E1 F3
G1 G2 G3 G4
```

Lembre-se de usar a classe `RunProject1.java` que lhe é fornecida conjuntamente com os ficheiros de *input* e de *output* esperados, bem como a classe de testes, para aferir (em parte) a correção do seu projeto.

O que entregar

Deve criar o ficheiro `P1fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros:

`TUTEDECC.java` e `RunProject1.java`

ATENÇÃO: Antes de submeter o trabalho, verifique que documentou o seu projeto (incluindo `@author` com o seu número de aluno).

Importante

O facto das vossas classes passarem nos testes efetuados não significa que a classe esteja 100% correta. Há testes e verificações que não são feitas de propósito, de modo a incentivar os alunos a irem à procura de pontos de eventuais falhas no código. Além disso, os pesos para a avaliação dados a cada um dos testes pode ser diferente.