

Projeto 1



Isabel Nunes

Unidade Curricular de
Laboratório de Programação

2021/2022

Objetivos

- Leitura e escrita de ficheiros;
- Manipulação de *strings*.

Antes de começar, algumas informações úteis

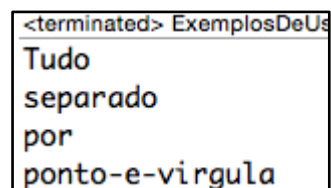
String e StringBuilder

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 e também as secções 7.2, 8.3, 8.4 e 8.5 do livro online em <http://www.di.fc.ul.pt/~in/PCOlivro/PCOonline/Index.htm>.

Relembrando: Em algumas classes, como por exemplo na classe `String`, existem métodos (por exemplo `split`) que permitem que, dado um **padrao** 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);  
        }  
    }  
}
```



O argumento que o método `split` recebe representa um padrão que é usado como separador dos elementos da `String`.

Se só nos interessa como separador um espaço, podemos usar:

```
linha.split(" ")
```

Se as palavras de interesse também pudessem estar separadas por pontos, por vírgulas e por ponto-e-vírgula, por exemplo, deveríamos ter usado:

```
linha.split("[ ,.;]")
```

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

Relembrar leitura e escrita de/em ficheiros

Neste projeto vai ter que ler as linhas de um ficheiro de texto e escrever outras linhas noutra ficheiro de texto.

O excerto de programa seguinte é um exemplo simples de leitura de um ficheiro que contém três linhas: uma linha com uma frase, outra com um inteiro e outra com uma frase.

```
public class LeituraLinhasFicheiro {  
  
    public static void main(String args[])  
        throws FileNotFoundException, IOException{  
  
        Scanner in = new Scanner (new File("meuFich.txt"));  
        PrintWriter out = new PrintWriter("teuFich.txt");  
  
        String nome = in.nextLine();  
        int idade = in.nextInt();  
        in.nextLine();      // para consumir a mudança de linha anterior  
        String texto = in.nextLine();  
  
        out.println(nome + " tem " + idade + " anos e diz o seguinte:");  
        out.println(texto.toLowerCase());  
  
        in.close();  
        out.close();  
    }  
}
```

Ciclos infinitos e escrita em ficheiros

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

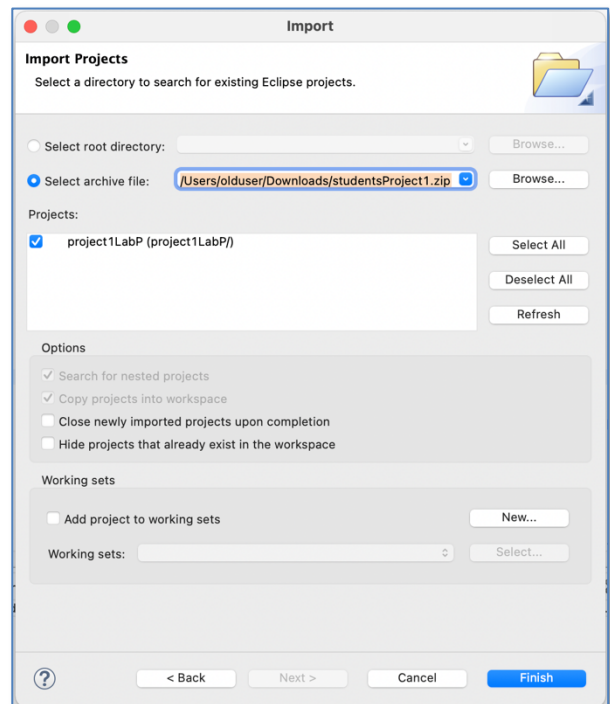
Isto já aconteceu a alguns alunos. Para evitar esta situação:

- Enquanto estiver a desenvolver o método escreva para o `System.out`;
- Se houver um ciclo infinito, rapidamente se aperceberá disso e poderá terminar a execução (basta clicar no quadrado encarnado na janela da consola);
- Quando tiver a certeza que o método já está correto, altera a escrita para ficheiro, usando então um objeto `PrintWriter`.

Vamos finalmente começar!

Importar o projeto

- Fazer *download* do ficheiro `studentsProject1.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 `studentsProject1.zip` e carregar em **Open**
 - depois de carregar em “Finish”, verá aparecer o projeto `project1LabP` 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 `project1LabP` que contém:

- Na pasta `src`, a classe `RunProject1` e 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 `OutputFiles.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.

Contexto do problema

Você foi contratado para construir um programa que permita codificar e decodificar mensagens alfabéticas aplicando a técnica descrita de seguida.

As mensagens de texto (somente letras) originais são escritas no alfabeto *standard* inglês (26 letras), usando só maiúsculas. Cada palavra de **k** carateres é codificada numa palavra cujo comprimento tem entre **k** e **k*3** carateres.

A codificação é feita com a ajuda de uma *chave de codificação*. Esta chave é composta de:

- uma sequência ordenada de **L** letras do alfabeto, chamada *chave alfabética* e
- um inteiro **N** no intervalo [1..25], chamado *chave numérica*.

As regras de codificação são as seguintes:

- *Regra 1:* Cada palavra do texto original é codificada separadamente; o espaço entre as palavras é preservado; cada palavra é codificada da esquerda para a direita;
- *Regra 2:* Uma letra **p** da mensagem de texto original que não pertença a **L** é codificada para uma letra **c** = *cod*(**p**) onde *cod* é uma função definida mais abaixo;
- *Regra 3:* Uma letra **p** da mensagem de texto original que pertença a **L** é codificada para uma *string* de 3 letras: a **m**-ésima letra de **L**, seguida de uma letra **c** = *cod*(**p**), seguida da (**m+1**)-ésima letra de **L**.

Para cada mensagem, o valor inicial de **m** é 1 e o seu valor é incrementado de uma unidade de cada vez que a regra 3 é aplicada; a aritmética é feita como se **L** fosse circular.

A função *cod* traduz cada letra para a letra **N** posições mais à frente no alfabeto (a aritmética é feita como se o alfabeto fosse circular; por exemplo, se **N** é 2, então *cod*(A) = C, *cod*(B) = D, ..., *cod*(X) = Z, *cod*(Y) = A, *cod*(Z) = B).

O que fazer então?

Deve desenvolver a classe `CoderDecoder` com os métodos públicos descritos de seguida.

Note que os parâmetros do tipo *String* identificam os **nomes** dos ficheiros.

- `void codify (String inFile, String outFile) throws FileNotFoundException, IOException` que lê do ficheiro de nome `inFile` a chave alfabética, a chave numérica, o número de mensagens que é preciso codificar e as mensagens (uma por linha), e escreve no ficheiro de texto de

nome `outFile` a codificação dessas linhas de acordo com o que foi explicado em cima.

- `void decodify (String inFile, String outFile) throws FileNotFoundException, IOException` que lê do ficheiro de nome `inFile` a chave alfabética, a chave numérica, o número de mensagens que é preciso decodificar e as mensagens (uma por linha), e escreve no ficheiro de texto de nome `outFile` a decodificação dessas linhas de acordo com o que foi explicado em cima. Caso uma linha não seja a codificação correta de nenhuma mensagem original, deverá ser escrita a frase “Error in codification” na linha correspondente.

Deverá criar métodos privados auxiliares de modo a estruturar bem as suas soluções.

FICHEIROS DE *INPUT*

Tanto os ficheiros contendo as mensagens originais (*input* para o método `codify`) como os ficheiros contendo mensagens já codificadas (*input* para o método `decodify`), têm a chave alfabética na primeira linha, a chave numérica na segunda linha e o número de mensagens a tratar na terceira linha. As linhas que se seguem (tantas quantas o número lido na terceira linha) contêm as mensagens a tratar.

Cada mensagem contém somente letras maiúsculas do alfabeto inglês *standard* (26 letras) e um espaço entre palavras.

FICHEIROS DE *OUTPUT*

Em ambos os métodos – `codify` e `decodify` – as três primeiras linhas do ficheiro de *input* deverão ser copiadas para o ficheiro de *output*. De seguida:

- No caso do método `codify`, deverá também ser escrito no ficheiro de *output* o resultado da codificação das várias mensagens do ficheiro de *input*.
- No caso do método `decodify`, deverá também ser escrito no ficheiro de *output* o resultado da decodificação das várias mensagens do ficheiro de *input*; uma mensagem de *input* que não seja a codificação correta de nenhuma mensagem original deverá dar origem à frase “Error in codification” na respetiva linha no ficheiro de *output*.

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**).

ATENÇÃO 3: Depois de ler o segundo valor inteiro do ficheiro deverá invocar o método `nextLine()` sobre o `Scanner` de modo a consumir o fim de linha que segue esse inteiro. Só depois deverá ler as linhas que se seguem. Se não fizer isso, o programa vai “ignorar” a primeira linha de mensagem após esse segundo inteiro.

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.

Antes de entregar

Antes de entregar, certifique-se da correção da formatação e inclua comentários *javadoc* para todos os métodos.

Inclua na classe `CoderDecoder` uma linha contendo a *tag* `@author` onde deve constar o seu nome e número de aluno.

O que entregar

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

`CoderDecoder.java` e `RunProject1.java`

Importante

O facto das classes dos alunos passarem nos testes efetuados, não significa que estejam 100% corretas. 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.