INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS

Campus Ouro Branco

Trabalho prático 2

Matheus Peixoto Ribeiro Vieira

Turma: INFO 3

Ouro Branco - MG

10/07/2021

Introdução

O trabalho prático 2 consiste em compreender mais sobre o acesso de arquivos por meio do Java e o uso de Regex.

O programa possui toda a parte de interação com usuário feita pelo terminal, onde o usuário utiliza seu teclado para ter acesso às áreas de informações, digitando, no menu principal, os números de 1 a 3, sendo o último para encerrar o programa.

Dentre as funcionalidades, o usuário pode encontrar a palavra mais comum com uma certa quantidade de caracteres ou encontrar em quais arquivos e em quais linhas estão uma palavra específica.

Descrição do desenvolvimento

O desenvolvimento do software foi feito no Netbeans 8.2 a partir da linguagem Java, com todo o código sendo escrito na classe TrabalhoPratico2.

O que foi solicitado para o programa eram duas funções principais, sendo elas, a procura pela palavra com uma quantidade de caracteres limitados que mais se repete em todos os 37 arquivos disponibilizados e uma varredura em todos os arquivos a procura de uma palavra que o usuário solicitou, tendo de ser buscada em todos arquivos e tendo suas respectivas linhas exibidas.

No início do método main há a instanciação de dois Scanners, um para texto e um para números. Sendo o primeiro rapidamente utilizado para solicitar ao usuário que ele informe o diretório onde os arquivos se encontram. E, caso o usuário passe um caminho inexistente, o programa fica num loop e uma mensagem é mostrada ao usuário informando-o do erro. Mas caso o programa encontre o diretório corretamente, todos os arquivos entrarão num vetor File denominado arquivos.

Após isso, o programa passa para um outro loop, mas dessa vez o usuário deve escolher qual menu ele deseja ir. Para entrar na procura de palavra mais repetitiva com uma certa quantidade de caracteres, ele digita 1 e o Scanner de números faz a leitura. Para ir para a busca de palavras, ele digita 2. E para finalizar o código, digita 3, que irá para um break, interrompendo o loop e indo ao fim do programa. Caso o usuário digite uma opção de menu inexistente, ele continuará no loop até informar uma opção verdadeira.

Na opção de menu 1, o usuário irá para o método **palavraMaisComumCaracterLimitado(File arquivos[]),** onde ele digitará uma quantidade de caracteres para encontrar a palavra mais comum com essa quantidade. Após isso, é instanciado um ArrayList palavrasEncontradas que receberá todas as palavras que o programa encontrar.

Então é iniciado um for com limite do tamanho do vetor arquivos. É aberto um try catch onde um FileReader e um BufferedReader são iniciados, além de uma String linha. Após isso o programa entra em um do while que durará até uma linha null ser encontrada, representando o fim do arquivo.

Dentro desse loop, as linhas são lidas e são divididas por um split com um espaço em branco delimitando a separação. Todas as palavras são enviadas

para um vetor palavras, que tem seu tamanho usado em um for que procura por palavras com o tamanho solicitado a partir de um REGEX de caracteres de A-Za-z, com número de caracteres limitados pelo usuário. Caso a palavra da vez satisfaça essa verificação, ela é mandada para o ArrayList palavrasEncontradas.

Após isso os arquivos e bufferes são fechados e o loop volta ao inicio até que todos os arquivos sejam verificados.

Depois da finalização do loop, o vetor palavrasEncontradas passa por um sort, a fim de ser organizado em ordem alfabética para facilitar verificações posteriores.

Os ArrayList String palavrasRepetidas e Integer vezesRepetidas são criados, assim como a variável inteira repeticoes e a String palavraVerificar, que recebe a primeira posição do vetor palavrasEncontradas.

Assim, começa um for do tamanho do vetor palavrasEncontradas menos 1, já que a verificação de palavras iguais é feita a partir da posição i do vetor palavrasEncontradas mais a próxima palavra. Se forem iguais, a variável repeticoes tem seu valor acrescido de um. Se não, o vetor palavrasRepetidas recebe a palavra verificada e o vetor vezesRepetidas recebe repeticoes+1, sendo, na mesma posição que o vetor anterior, essa variável zerada e palavraVerificar recebe a próxima palavra do palavrasEncontradas.

Ao sair do for, há uma verificação simples de qual palavra foi repetida mais vezes e ela é impressa, assim como a sua quantidade de vezes que aparece nos arquivos.

Caso a opção escolhida seja a 2, o usuário irá para o método **buscarPalavra(File arquivos[])**, onde tem a palavra que deseja procurar lida pelo sistema, que começa um for com o tamanho do vetor arquivos, sendo que todo o resto do método está dentro desse loop de repetição, que passa de arquivo em arquivo.

Então um ArrayList Integer linhas é criado para armazenar o número das linhas que as palavras se encontram. Depois um FileReader, um BufferedReader, e uma String linha são criados para ler as linhas do arquivo.

É iniciado um do while que lerá os arquivos até encontrar uma linha nula, que significa o fim do mesmo.

Dentro desse laço de repetição, as linhas são separadas pelo split nos espaços em brancos e todas as palavras são adicionadas no vetor palavrasNaLinha, que é o limite para um for seguinte, que verificar todas as palavras a partir de 8 possibilidades a fim de encontrar as mais variadas possibilidades de escritas. Sendo usado assim, oito combinações de REGEX diferentes, como segue na imagem abaixo.

```
if(palavrasNaLinha[cont].contains(palavraPesquisar) && palavrasNaLinha[cont].matches("[A-Za-z]{"+palavraPesquisar.length()+"}") || //Ex: Charles
        palavrasNaLinha[cont].matches("[\\W][A-Za-z]{"+palavraPesquisar.length()+"}") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: (Charles
        palavrasNaLinha[cont].matches("[\\W][A-Za-z]{"+palavraPesquisar.length()+"}[\\W][\\D][\\W]") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: "Charles's.
        palavrasNaLinha[cont].matches("[\\W][A-Za-z]{"+palavraPesquisar.length()+"}[\\W][\\D]") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: "Charles's
        palavrasNaLinha[cont].matches("[\\W][A-Za-z]{"+palavraPesquisar.length()+"}[\\W]") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: "Charles,
        palavrasNaLinha[cont].matches("[A-Za-z]{"+palavraPesquisar.length()+"}[\\W][\\D][\\W]") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: Charles's.
        palavrasNaLinha[cont].matches("[A-Za-z]{"+palavraPesquisar.length()+"}[\\W][\\D]") && palavrasNaLinha[cont].contains(palavraPesquisar) || //Ex: Charles's
        palavrasNaLinha[cont].matches("[A-Za-z]{"+palavraPesquisar.length()+"}[\\W]") && palavrasNaLinha[cont].contains(palavraPesquisar)){ //Ex: Charles,
    linhas.add(numeroDaLinha);
}
```

Tomemos por exemplo o nome Charles, que pode estar escrito das seguintes formas “Charles”, “(Charles”, “(Charles’s.”, “Charles’s”, “”Charles,””, “Charles’s.”, “Charles’s”, “Charles,”.

Como a palavra Charles se adeque a alguma dessas variações, logo ela será inserida na lista e sua linha será computada no ArrayList linhas.

Somente o matches com o REGEX se adequa a esse caso de verificação, já que o equals ignoraria todas essas possibilidades de variação, e o contains serviria somente para algumas palavras, gerando problema para as outras., um exemplo é a palavra “the”, que seria encontrada facilmente, porém seria listado, também, quando ela aparece em “therefore”, “them”, etc.

Vale ressaltar que a palavraPesquisar e a palavra atual do vetor palavrasNaLinha se tornam todas em caixa alta, para que não haja problemas de verificação caso no texto a palavra esteja escrita de forma diferente, um exemplo é passar a palavra “the” e no texto haver as variações “THE” ou “The”.

Ao sair do loop, o vetor linhas é verificado e, se ele for maior do que zero, será mostrado em quais arquivos a palavra aparece e em quais linhas

Por fim, a opção três leva ao encerramento do programa.

Conclusão

Em geral, esse trabalho foi muito tranquilo de ser realizado. Sendo que eu tive poucos erros, porém que demoraram um pouco para serem concertados. Além disso, demorei um pouco para pensar em como verificar as palavras no método de encontrar as palavras que mais se repetem. E, no método 2, demorei para entender quais tipos de variações a palavra poderia ter, sendo que tive que abrir um arquivo e procurar pela palavra em todas as linhas, verificando a linha em que ela se encontrava e se o programa havia reconhecido.

Bibliografia

- JAVATPOINT. **How to Sort ArrayList in Java**. Disponível em: https://www.javatpoint.com/how-to-sort-arraylist-in-java. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Aula 10 - Strings (Meta caracteres)**. Disponível em: https://www.youtube.com/watch?v=_1ADXDoUBzM&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Aula 11 - Strings (Exemplos quantificadores)**. Disponível em: https://www.youtube.com/watch?v=mmDcMQ2gnVk&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Aula 7 - Strings (Split)**. Disponível em: https://www.youtube.com/watch?v=-hK_3P9YrHw&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Aula 8 - String (Exemplo indexOf)**. Disponível em: https://www.youtube.com/watch?v=GmC3u_kowbw&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Aula 9 - String (criando REGEX)**. Disponível em: https://www.youtube.com/watch?v=AWnWNuIcAvg&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.
- SILVA, Saulo Henrique Cabral. **Processamento de arquivos (parte 8)**. Disponível em: https://www.youtube.com/watch?v=W_tLWZQYxBU&ab_channel=SauloHenriqueCabralSilva. Acesso em: 22 jun. 2021.