UnB - Universidade de Brasilia
FGA - Faculdade do Gama
TPPE - Técnicas de Programação em Plataformas Emergentes

Trabalho Prático 1 - Desenvolvimento Orientado a Testes (TDD)
Desenvolvimento de parser de resultados de análises de confiabilidade de linhas de produto de software.

1) O Parser:

O programa a ser desenvolvido consiste em um parser de arquivos-texto contendo valores de tempos de análise e de execução de uma ferramenta que avalia a confiabilidade de programas de software. Para cada "versão" do software avaliada (também chamada de evolução) é realizada uma quantidade de análises de confiabilidade e, para cada iteração, são anotados os tempo de análise e o tempo total de execução da ferramenta. Por exemplo, um software com 20 evoluções analisadas 10 vezes resultará em um total de 200 avaliações realizadas, cada avaliação com seus tempos anotados.

Esses valores estão hoje armazenados em dois arquivos-texto: **analysisTime.out** e **analysisMemory.out**. Estruturalmente não há diferenças entre os arquivos sendo o primeiro arquivo o registro de tempo de avaliação de cada versão e o segundo arquivo o consumo total de memória pela ferramenta de análise durante sua execução. Cada evolução está destacada através de uma "String" no arquivo, seguida pelos valores de tempo anotados em milisegundos (a quantidade de valores denota o número de iterações realizadas para aquela evolução) ou memória em Mbytes. Em seguida está apresentado um pequeno pedaço de um desses arquivos.

```
---------- Evolution 1 ----------
456
782
755
729
761
751
869
823
847
730
---------- Evolution 2 ----------
523
861
770
783
720
813
750
726
770
771
---------- Evolution 3 ----------
458
973
803
753
758
783
749
```

```
794
769
```

É comum que os valores registrados nesses arquivos cresça significativamente tanto em número de evoluções quanto em número de iterações (análises) realizadas. Quando isso acontece, a análise desses tempos torna-se mais difícil, principalmente para realizar análises visuais e estatísticas dos dados. Atualmente é feita uma série de transformações nesses arquivos como pré-processamento para a fase de análise. Essas transformações são feitas em scripts *bash* e estão muito sensíveis a pequenas variações nas estruturas desses arquivos.

Portanto deseja-se implementar uma ferramenta que seja capaz de ler esses valores dispostos sequencialmente nos arquivos **analysisTime.out** e **analysisMemory.out** e persisti-los em arquivos texto com campos separados por valor. Muito parecido com arquivos CSV (em que campos são separados por vírgulas, do inglês *comma-separated values*), esses arquivos terão seus valores separados por caractere(s) definidos pelo usuário. A depender da quantidade de valores dos arquivos o usuário escolherá se as sequências de valores serão dispostas em linhas ou em colunas. Considerando o trecho de arquivo apresentado anteriormente, se a opção do usuário for apresentar as sequencias em **linhas** e **';'** como delimitador de campos, o arquivo texto resultante será o seguinte:

```
1;456;782;755;729;761;751;869;823;847;730
2;523;861;770;783;720;813;750;726;770;771
3;458;973;753;758;783;749;794;769
```

No caso em que o usuário escolher apresentar as sequências em **colunas** e ';' como caractere separador de campos, o arquivo texto resultante será o seguinte:

```
1;2;3
456;523;458
782;861;973
755;770;753
729;783;758
761;720;783
751;813;749
869;750;794
823;726;769
847;770
730;771
```

Notem que não há a necessidade que todas as evoluções tenham o mesmo número de iterações. No exemplo acima, a evolução três tem dois valores a menos que as evoluções um e dois.

2) O aplicativo

O aplicativo a ser desenvolvido como tema desse trabalho consiste em uma ferramenta para realizar a pré-análise dos dados dos arquivos **analysisTime.out** e **analysisMemory.out**. O aplicativo deverá apresentar as seguintes funcionalidades:

- Definições das configurações das pré-análises:

- Leitura do arquivo de entrada (**analysisTime.out** ou **analysisMemory.out**)
  - Se o arquivo não existir no diretório, uma exceção do tipo **ArquivoNaoEncontradoException**, informando o nome do arquivo, deverá ser lançada e capturada pelo programa.
- Definição do delimitador de campo a ser utilizado no arquivo de saída. Considere o delimitador como sendo um caractere apenas.
  - Se o delimitador informado for uma "string", uma exceção **DelimitadorInvalidoException** informando o erro deverá ser lançada e capturada pelo programa. Atenção: caracteres com o escape '\' são considerados um caractere apenas. Exemplos: \r, \n, \t .
- Definição do caminho do arquivo de saída.
  - Caso não haja permissão de escrita no caminho de saída, uma exceção do tipo **EscritaNãoPermitidaException** deverá ser lançada e capturada pela aplicaçãp.
- Definição do formato de saída das sequencias de valores.
  - O usuário deverá ser capaz de indicar se quer que as sequencias de tempos para cada evolução seja apresentada em linhas ou em colunas, conforme exemplos apresentados anteriormente.
- Parser dos valores dos arquivos:
  - A aplicação deverá ser capaz de identificar automaticamente, i.e. **sem intervenção do usuário**, quantas evoluções foram analisadas e quantas análises foram feitas para cada evolução.
- Escrever arquivo resposta:
  - Após o parser dos arquivos de entrada o aplicativo deverá ser capaz de transformar para a forma de saída indicada pelo usuário e escrevê-la em arquivo texto com os seguintes nomes **analysisTimeTab.out** no caso do arquivo de entrada ser o arquivo analysisTime.out ou **analysisMemoryTab.out** no caso do arquivo de entrada ser totalTime.out

A ferramenta deverá ser implementada utilizando a técnica de **Desenvolvimento Orientado a Testes (TDD)**, em linguagem Java, utilizando o framework JUnit 4 ou superior. A implementação utilizando TDD deverá ser evidenciada através do histórico de *commits* do projeto. Recomenda-se que cada uma das funcionalidades apresentadas acima seja desenvolvida em sua própria *branch*, cujos *commits* apresentarão as técnicas de TDD que foram utilizadas (falsificação, duplicação, triangulação). Ao final da implementação de cada funcionalidade, sua respectiva branch será fundida (*merge*) na branch **Dev**.

O trabalho deverá ser realizado em grupos de até três componentes. Os nomes e matrículas dos componentes deverão ser apresentados no arquivo README.md do repositório. A avaliação da participação dos membros no trabalho será realizada pela análise de autoria dos commits.

Cada grupo deverá enviar email para o professor indicando o nome dos componentes do grupo e o endereço do repositório Github do trabalho.

Valor do trabalho: 40% da nota final.
Data de entrega: 18 de outubro de 2021.
Forma de entrega: via repositório Github.