# Trabalho – 1

O objetivo deste trabalho é construir um programa capaz de avaliar se a estrutura de um arquivo HTML está correta. Um arquivo HTML é um arquivo utilizado para criar documentos para a web. Trata-se de um arquivo em formato texto constituído de comandos denominados de *tag*, utilizados para formatar o texto a ser exibido no navegador de internet. Veja um exemplo de conteúdo de um arquivo HTML simples:

```
<html>
<body>
<h1>Aqui cabeçalho do arquivo</h1>
<p>Meu parágrafo da página web.</p>
<p>Meu segundo parágrafo.</p>
</body>
</html>
```

Neste arquivo, as *tags* (comandos) são: **`html`**, **`body`**, **`h1`** e **`p`**. Observe a terceira linha do arquivo que é constituído da *tag* **`h1`**. Esta linha possui uma *tag de início*, chamada de **`<h1>`** e uma *tag final*, chamada de **`</h1>`**. Normalmente as *tags* do arquivo HTML estão em pares, como exposto neste exemplo, sendo que a *tag final* é igual à *tag de início*, porém possui um caractere "/" após o caractere "<". O mesmo ocorre com as outras *tags* (**`html`**, **`body`** e **`p`**).

É importante observar também que uma *tag final* somente pode ser inserida quando não houver outra nova *tag de início* que ainda não possui sua *tag final*. Isto é, a *tag final* **`</body>`** não pode ser colocada antes de **`</h1>`**, já que a *tag de início* **`<h1>`** deve ser finalizada antes.

Um arquivo HTML pode ser constituído de diversas *tags*, além de **`html`**, **`body`**, **`h1`** e **`p`** (estes são apenas alguns exemplos de *tags*). Um comando (*tag*) é uma sequência de caracteres/algarismos precedida de "<".

O seu programa deverá permitir receber um arquivo a ser analisado e o programa deverá verificar se a estrutura de *tag inicial* e *tag final* estão corretas. Para resolver este problema, implemente uma pilha de *tags*: quando você extrair do arquivo uma *tag de início,* coloque-o numa pilha. Quando ler uma *tag final*, retire um dado da pilha e verifique se o dado retirado corresponde a *tag de início* da *tag final*, isto é, verifique se forma um par. Caso não formar, há um erro de formatação de *tags*.

Interpreta-se que o arquivo está bem formatado quando todas as *tags de início* possuem suas respectivas *tags finais*. Caso a estrutura do arquivo estiver mal formatada, seu programa deverá informar ao usuário que existem *tags de início* sem *tag final*.

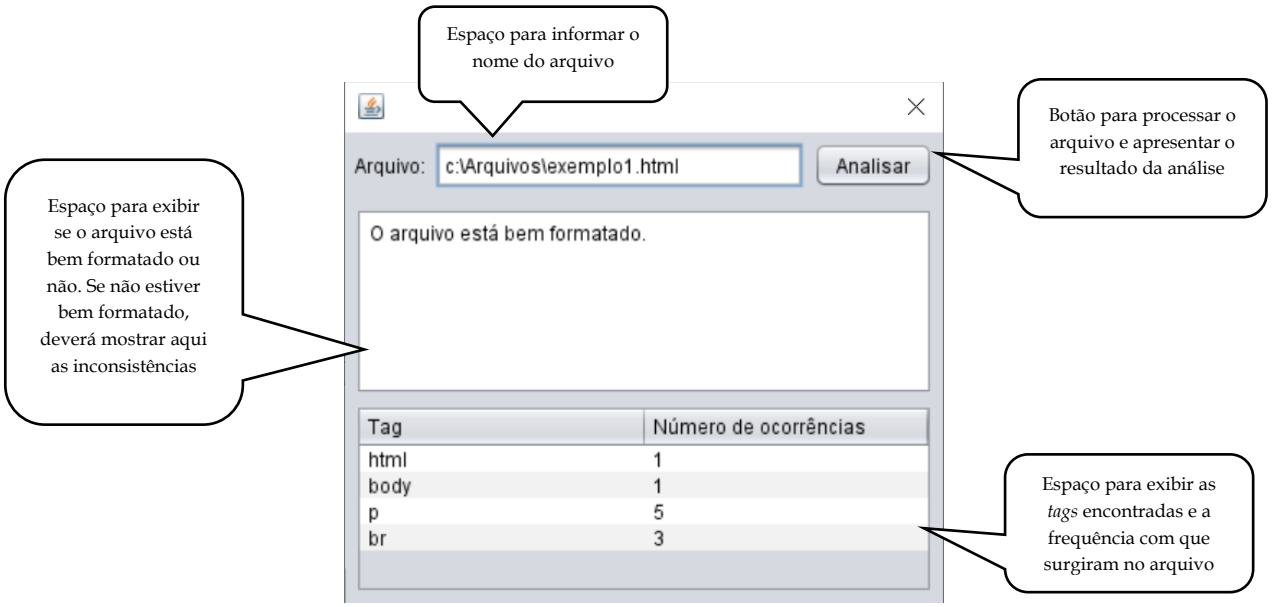
Existe algumas exceções para se tratar neste programa:

a) Linhas em branco devem ser desprezadas;
b) O tratamento das *tags* deve ser *case insentitive*;
c) Uma *tag* pode ser constituída de atributos, como o exemplo abaixo:
   **`<a href="http://www.w3schools.com">Isto é um link</a>`**
   Neste exemplo, a *tag de início* é **`<a>`** e ela possui o atributo **`href`**. Os atributos não são importantes para nossa análise, mas deve-se reconhecer que a tag é **`<a>`** e não **`<a href...>`**.
d) Existe um conjunto de *tags*, denominadas de *singleton tags*. Estas *tags* não têm uma *tag final* e portanto, não deve ser validado se a *tag* possui uma *tag final*. Considere que as *singleton tags* são estas: **`meta`**, **`base`**, **`br`**, **`col`**, **`command`**, **`embed`**, **`hr`**, **`img`**, **`input`**, **`link`**, **`param`**, **`source`** e **`!DOCTYPE`**. Todas estas *tags* podem ou não ser utilizadas com atributos.

Quando um arquivo estiver corretamente formatado você também deverá apresentar a relação de *tags* utilizadas no arquivo, bem como a quantidade de vezes que apareceram no arquivo. No exemplo anterior, o programa deveria informar que a tag **`html`** foi utilizada uma vez e a *tag* **`p`** foi utilizada 2 vezes, por exemplo.

Os requisitos deste trabalho são:

1. Construir diagrama de classes da solução;
2. O programa deverá avaliar a estrutura do arquivo fornecido pelo usuário e indicar se o arquivo está bem formatado ou não, considerando os pares *tag de início/fim*, *singleton tags* e *tags* com atributos;
3. Se o arquivo estiver bem formatado, deverá ser apresentado na tela uma relação das *tags* encontradas bem como a frequência de cada uma. As *singleton tags* também devem ser computadas. Os dados devem ser ordenados por nome de tag.
4. Se o arquivo não estiver bem formatado, o programa deverá indicar qual a razão, que poderá ser:
   4.1. Foi encontrada uma *tag final* inesperada (aguardava-se determinada *tag final* mas foi encontrada outra). Deve-se indicar qual a *tag final* encontrada e qual a *tag final* esperada;
   4.2. Faltam *tags* finais. Neste caso, apresentar quais as *tags finais* esperadas mas não encontradas.
5. O programa deve ter uma tela gráfica semelhante à tela abaixo:




O trabalho deve ser feito em dupla. Deve ser submetido no AVA até 17/11/2023, num arquivo compactado com o nome "trabalho.zip", contendo todos os arquivos fontes e o diagrama de classes em formato .jpg. O projeto deve possuir somente uma classe com o método **`main()`**. Acrescente também um arquivo chamado **`readme`** e escreva o nome completo dos participantes da equipe.

No arquivo "trabalho.zip" acrescente também o programa executável no formato .jar. Chame-o de "Trabalho.jar".

A solução que for implementada não pode utilizar nenhuma classe de coleções da API Java (**`ArrayList`**, **`HashMap`**, etc). Utilize apenas as estruturas de dados implementadas nos exercícios desta disciplina. As implementações que foram construídas não deveriam precisar de adaptação para atender a este trabalho. Também não podem ser utilizadas classes que não sejam nativas do pacote Java.