En esta tarea, usarás las implementaciones de pila y cola de la API de Colecciones de Java para construir un programa que determine si una página HTML está bien formada.

Al completar esta tarea, lograrás:

- Familiarizarte con los métodos de la clase **java.util.Stack** y la interfaz **java.util.Queue**.
- Trabajar con un tipo de dato abstracto (específicamente, colas) utilizando solo la interfaz de una implementación.
- Aplicar lo que has aprendido sobre cómo funcionan las pilas y colas.

Nota sobre depuración/errores:

Si te encuentras con errores o no entiendes el resultado que te da Codio, publica en el Foro de Discusión, y un asistente de enseñanza (TA) te ayudará. ¡Por favor, NO envíes correos a Codio, ¡ya que no revisarán los errores que estés obteniendo!

Antecedentes:

Las páginas web están escritas en Lenguaje de Marcado de Hipertexto (HTML). Un archivo HTML se compone de texto rodeado por etiquetas, donde las etiquetas "marcan" el texto especificando su formato, diseño u otra información. Las etiquetas también pueden estar anidadas. Aquí tienes un ejemplo simple, con las etiquetas resaltadas en negrita:

```
<html>
    <head>
        <title>Sample HTML page</title>
    </head>
    <body>
        This is some <b>HTML text!</b>
    </body>
</html>
```

El significado exacto de las etiquetas no es importante. Sin embargo, las etiquetas como `<body>` y `<b>` se conocen como "etiquetas de apertura" porque indican el inicio de algún formato, y las etiquetas como `</body>` (con la barra diagonal antes de la palabra) se conocen como "etiquetas de cierre" porque indican el final del formato.

En teoría (aunque no a menudo en la práctica), un HTML bien formado requiere que las etiquetas estén "balanceadas", es decir, que las etiquetas de apertura tengan su correspondiente etiqueta de cierre en el orden correcto.

Por ejemplo, si ignoramos los espacios en blanco y el texto entre las etiquetas, obtenemos esto:

<html><head><title></title></head><body></body></html>

Observa que hay cierta simetría en las etiquetas HTML, ya que cada vez que cerramos una etiqueta, coincide con la última etiqueta de apertura (no cerrada) más reciente.

Por ejemplo, si destacamos las etiquetas “title”, vemos que una etiqueta de cierre coincide con la última etiqueta de apertura:

<html><head><title></title></head><body></body></html>

Y en este caso, la etiqueta de cierre “body” coincide con la etiqueta de apertura “body”, que es la etiqueta de apertura más reciente que aún no se ha cerrado (ya que la etiqueta “b” ya está cerrada):

<html><head><title></title></head><body></body></html>

Algunas etiquetas HTML se "cierran por sí solas" y no dependen de una etiqueta de cierre correspondiente. Por ejemplo, aquí la etiqueta “br” se cierra sola:

<html><head>head><body>
</body></html>

Una etiqueta autoconclusiva es aquella que termina con el carácter de barra diagonal (`/`), a diferencia de una etiqueta de cierre, que comienza con una barra diagonal.

¡Es fácil cometer errores en el código HTML! Lo más común es que las personas olviden cerrar las etiquetas o cierren las etiquetas anidadas en el orden incorrecto, por ejemplo, algo como esto:

<html><head><title></title><body></body></html>

En este caso, falta una etiqueta de cierre </head>, y la etiqueta </body> está cerrada en el orden incorrecto: debería cerrarse después de la etiqueta de cierre .

En esta tarea, escribirás un método que determine si un archivo HTML está bien formado usando una pila (`stack`). Cada vez que tu código encuentre una etiqueta de apertura, debería empujarla (`push`) en la pila; cuando encuentre una etiqueta de cierre, debería sacar (`pop`) la etiqueta de la parte superior de la pila, y si no coinciden, sabrás que el archivo no está bien formado. A continuación, se proporcionan más ejemplos y explicaciones.

Comenzando

Descarga los archivos `htmlreader.java` y `htmltag.java`, que contienen el código que puedes utilizar en esta tarea. No deberías cambiar ninguna de estas implementaciones para esta tarea.

`HtmlTag.java` representa la información sobre una sola etiqueta HTML. Métodos que pueden ser útiles para ti:

- `getElement()`**: Obtiene el nombre del elemento (String) especificado en esta etiqueta.
- `isOpenTag()`**: Verifica si esta es una etiqueta de apertura. Si la etiqueta es de cierre o autoconclusiva (por ejemplo, `<br/>` es una etiqueta de salto de línea que no necesita ningún texto adicional), `isOpenTag` devolverá `false`.
- `isSelfClosing()`**: Verifica si una etiqueta se cierra sola (por ejemplo, `<br/>`).
- `matches(HtmlTag other)`**: Verifica si una etiqueta `HtmlTag` es la etiqueta de apertura/cierre correspondiente a otra etiqueta (por ejemplo, `<b>` y `</b>` o viceversa).

En el archivo `HtmlReader.java`, encontrarás un método llamado `getTagsFromFile` que lee la ruta de un archivo HTML y lo separa en tokens. La salida es una representación del archivo HTML como una cola (`Queue`) de `HtmlTags` en el orden en que se encontraron. Puedes editar este código si lo deseas, pero no modifiques `HtmlTag.java`.

También descarga el archivo `HtmlValidator.java`, que contiene el método no implementado para el código que escribirás en esta tarea.

Actividad

En `HtmlValidator.java`, implementa el método `isValidHtml`. Este método debe tomar como entrada una cola (`Queue`) de `HtmlTags` y devolver una pila (`Stack`) de `HtmlTags` que verifique la corrección de la estructura de las etiquetas, según la especificación descrita a continuación.

Especificaciones del método:

HTML Bien Formado:

Si el archivo HTML está bien formado, el método debe devolver una pila vacía.

Ejemplo:

<html><body><h1>heading</h1> paragraph </body></html>

En este caso, las etiquetas de cierre coinciden con las etiquetas de apertura, por lo que el HTML es válido. Cuando llegues al final del archivo/cola, la pila estará vacía.

HTML No Bien Formado:

Si el archivo HTML no está bien formado, el método debe devolver la pila en su estado actual (es decir, con sus valores actuales) en el momento en que el código determinó que las etiquetas no estaban balanceadas.

Ejemplo 1: Etiquetas cerradas en orden incorrecto:

<html><body> Sentence here </body></html>

En este caso, debes empujar todas las etiquetas de apertura en la pila para que se vea así:

<body>

Al encontrar una etiqueta de cierre en la cola, querrás comprobar que la pila tiene la coincidencia correcta. La primera etiqueta de cierre que encuentras es `</p>`, pero la última etiqueta de apertura (en la parte superior de la pila) es `<b>`. Eso es incorrecto. Tan pronto como determines que el archivo HTML no es válido, **devuelve la pila de etiquetas de apertura** sin sacar la etiqueta de apertura no coincidente.

En este caso, la salida esperada sería una pila que contenga (de abajo hacia arriba): `<html><body><p><b>`.

Ejemplo #2: Etiqueta de cierre sin etiqueta de apertura

<html><body>Correct
Sentence

here </body></html>

En este caso, la primera etiqueta de cierre que encuentras (`</b>`) coincide con su etiqueta de apertura, pero la siguiente (`</div>`) no lo hace, por lo que la salida esperada sería una pila que contenga (de abajo hacia arriba): `<html><body>`.

Nota que la etiqueta `<br/>` es autocerrada y no debería ser colocada en la pila.

Ejemplo #3: Etiqueta de apertura nunca cerrada

<html><body>This is some text

En este caso, el método llega al final del archivo/cola y todavía hay elementos en la pila, ya que esas etiquetas de apertura nunca se cerraron. La salida esperada sería una pila que contenga (de abajo hacia arriba): `<html><body><b>`.

Ejemplo #4 (¡la parte complicada!): Etiqueta de cierre sin etiqueta de apertura, todo bien hasta entonces

<html><body> Hello, world! </body></html>

Esto es similar al Ejemplo #2, excepto que ahora, cuando encuentras la etiqueta de cierre que no tiene etiqueta de apertura, la pila está vacía, ya que todo antes de eso está emparejado. Sin embargo, devolver una pila vacía significa que el archivo está bien formateado. En este caso, debes devolver `null` para indicar que el archivo no está bien formateado. Piensa en cómo puedes diferenciar entre cuándo devolver `null` y cuándo devolver una pila vacía.

Por favor, no cambies la firma del método `isValid` (su lista de parámetros, nombre y tipo de valor de retorno). Además, no crees archivos `.java` adicionales para tu solución y no modifiques `HtmlTag.java`. Si necesitas clases adicionales, puedes definirlas en `HtmlValidator.java`. Por último, asegúrate de que tu clase `HtmlValidator` esté en el paquete predeterminado, es decir, que no haya una declaración de "package" en la parte superior del código fuente.

Consejos útiles

La documentación sobre los métodos en la clase `Stack` y la interfaz `Queue` en la versión más reciente de Java está disponible en:

- [Documentación de Stack] (https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html)
- [Documentación de Queue] (https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html)

Consulta esta documentación si necesitas ayuda para entender los métodos que tienes disponibles.

Ten en cuenta que tu método `HtmlValidator.isValidHtml` solo debe usar métodos de la interfaz `Queue`, aunque la `Queue` esté implementada usando un `LinkedList`.

Está bien si tu método `isValidHtml` modifica el contenido de la `Queue` que se pasa como entrada, por ejemplo, eliminando elementos.

Antes de enviar

Asegúrate de que:

- Tu clase `HtmlValidator` esté en el paquete predeterminado, es decir, que no haya una declaración de "package" en la parte superior del código fuente.
- Tu clase `HtmlValidator` compile y no hayas cambiado la firma del método `isValidHtml`.
- No hayas creado archivos `.java` adicionales y no hayas hecho cambios en `HtmlTag.java` (no necesitas enviar este archivo ni `HtmlReader.java`).

1. Descarga la distribución de JUnit en `junit-dist.jar`. Sigue estos pasos para agregar la biblioteca.

2. [Cómo agregar una ruta de construcción a una carpeta de clases en IntelliJ](https://intellij-support.jetbrains.com/hc/en-us/community/posts/360009909039-How-do-I-add-a-build-Path-to-a-class-folder-)
3. Descarga las pruebas en `homework2-tests.jar` y agrégalas a la ruta de construcción del proyecto de Eclipse como se indica arriba.
4. También descarga los archivos de entrada de prueba desde `homework2-files.zip`. Descomprime este archivo en tu computadora y copia los siete archivos `.html` en tu proyecto de IntelliJ; deberías poder arrastrarlos y soltarlos directamente en IntelliJ. Asegúrate de ponerlos en el directorio raíz de tu proyecto, al igual que con los dos archivos `.jar`.
5. Ahora ejecuta las pruebas haciendo clic derecho en `homework2-tests.jar` en IntelliJ para obtener el menú emergente/contextual y seleccionando "Run As -->" y luego "Java Application". Deberías ver las pruebas ejecutarse en la consola y te debería decir tu puntuación para esta tarea, o "¡Buen trabajo!" si tu puntuación sería del 100%.

Alternativamente, si deseas ejecutar el calificador automático desde la línea de comandos, coloca los dos archivos `.jar` y tus archivos `.class` para esta tarea en el mismo directorio junto con los archivos `.html` que descargaste, y ejecuta:

Mac/Linux: `java -cp .:junit-dist.jar:homework2-tests.jar Homework2Grader`

Windows: `java -cp .;junit-dist.jar;homework2-tests.jar Homework2Grader`

Esto agregará `junit-dist.jar` y `homework2-tests.jar` al classpath y luego ejecutará Java con `Homework2Grader` como la clase principal.