En esta tarea, utilizará las implementaciones de pila y cola de la API de colecciones de Java para crear un programa que determine si una página HTML está bien formateada.

Al completar esta tarea, usted:

- Familiarícese con los métodos de la clase java.util.Stack y la interfaz java.util.Queue
- Trabaje con un tipo de datos abstracto (específicamente, colas) usando solo la interfaz de una implementación
- Aplique lo que ha aprendido sobre cómo funcionan las pilas y las colas

Depuración/Nota de error:

Si se encuentra con errores/bugs/no entiende el resultado que Codio le está dando, publique en el foro de discusión y un TA lo ayudará. NO envíe un correo electrónico a Codio, ya que no revisarán ningún error que esté recibiendo.

Fondo

Las páginas web están escritas en lenguaje de marcado de hipertexto (HTML). Un archivo HTML se compone de texto rodeado de etiquetas, donde las etiquetas "marcan" el texto especificando su formato, diseño u otra información. Las etiquetas también se pueden anidar.

Aquí hay un ejemplo simple, con las etiquetas resaltadas en negrita:

<html>

<título><título>Muestra HTML página**</título></cabeza>**

<cuerpo>

Esto es tan yo**<segundo>**texto HTML**!**

</cuerpo>

</html>

Los significados exactos de las etiquetas no son importantes (aunque si desea obtener más información, puede inscribirse en nuestro curso "Programación para la Web con JavaScript" como parte de esta serie de edX), pero las etiquetas como**<cuerpo>**y**<segundo>**se conocen como "etiquetas abiertas" porque indican el inicio de algún formato y etiquetas como**</cuerpo>**y (con la barra inclinada delante de la palabra) se conocen como etiquetas de "cierre" porque indican el final del formato.

En teoría (aunque no a menudo en la práctica), el HTML bien formateado requiere que las etiquetas estén "equilibradas", es decir, que las etiquetas abiertas coincidan con su etiqueta de cierre correspondiente en el orden correcto.

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

<html><título><título></título></título><cuerpo></cuerpo></html>

Tenga en cuenta que existe cierta simetría en las etiquetas HTML, ya que cada vez que cerramos una etiqueta, coincide con la etiqueta abierta más reciente (no cerrada).

Por ejemplo, si resaltamos las etiquetas de "título", vemos que una etiqueta cerrada coincide con la última etiqueta abierta:

<html><cabeza>**<título></título>**</head><cuerpo></cuerpo></html>

Y en este caso, la etiqueta de "cuerpo" cerrada coincide con la etiqueta de "cuerpo" abierta, que es la etiqueta abierta más recientemente que aún no se ha cerrado (dado que la etiqueta "b" ya está cerrada):

<html><encabezado><título></título></encabezado>**<cuerpo>****</cuerpo>**</html>

Algunas etiquetas HTML son de "cierre automático" y no dependen de una etiqueta de cierre coincidente. Por ejemplo, aquí la etiqueta "br" se cierra sola:

```
<html><cabeza>cabeza><cuerpo><b><br/></b></cuerpo></html>
```

Una etiqueta de cierre automático es aquella que termina con el carácter de barra diagonal, a diferencia de una etiqueta de cierre, que comienza con uno.

¡Es fácil cometer errores en el código HTML! Por lo general, las personas se olvidan de cerrar las etiquetas o cierran las etiquetas anidadas en el orden incorrecto, por ejemplo, algo como esto:

```
<html><encabezado><título></título><cuerpo><b></cuerpo></b></html>
```

En este caso, no hay una etiqueta de "cabeza" de cierre y la etiqueta de "cuerpo" se cierra en el orden incorrecto: debe ir después de la etiqueta de "b" de cierre.

En esta tarea, escribirá un método que determina si un archivo HTML está bien formateado usando una pila. Cada vez que su código encuentra una etiqueta abierta, debe colocarla en la pila; cuando encuentra una etiqueta cercana, debe sacar la etiqueta de la parte superior de la pila y, si no coinciden, sabrá que el archivo no está bien formateado. Más ejemplos y explicaciones se proporcionan a continuación.

Empezando

Descargue htmlreader.java y htmltag.java, que contienen código que puede usar en esta tarea. No debe cambiar ninguna de estas implementaciones para esta tarea.

HTMLTag.javarepresenta información sobre una única etiqueta HTML. Métodos que te pueden ser útiles:

- **obtenerElemento()**Obtiene el nombre del elemento (String) especificado en esta etiqueta.
- **isOpenTag()**Comprueba si esta es la etiqueta de apertura. Si la etiqueta es la etiqueta de cierre o cierre automático (p.ej**
es una etiqueta de salto de línea que no necesita ningún texto adjunto),isOpenTag**regresará**FALSO**.
- **esSelfClosing()**Comprueba si una etiqueta se cierra sola (p. ej.**
**)

- **coincidencias (HtmlTag otro)**Comprueba si una HtmlTag**otro**es la etiqueta de apertura/ cierre correspondiente a sí misma (p. ej.**<segundo>**y**</segundo>**o viceversa).>

En**HtmlReader.java**encontrará un método llamado**getTagsFromFile**que lee la ruta a un archivo HTML y lo separa en tokens. El resultado es una representación del archivo HTML como una cola de*Etiquetas Html*en el orden en que fueron encontrados. Puede editar este código si lo desea, pero no lo modifique.*HTMLTag.java*.

También descargue htmlvalidator.java, que contiene el método no implementado para el código que escribirá en esta tarea.

Actividad

En**HtmlValidator.java**, implementar el**esValidHtml**método.**esValidHtml**debe tomar como entrada una cola de HtmlTags y devolver una pila de HtmlTags que verifique la corrección de la estructura de la etiqueta, de acuerdo con la especificación que se describe a continuación.

El método debe implementarse de la siguiente manera:

Si el archivo HTML está bien formateado, el método debería devolver una pila vacía. Por ejemplo:

<html><body><h1>título</h1> párrafo </body></html
>

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

Si el archivo HTML no está bien formateado, el método debería 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 equilibradas.

Aquí hay algunos casos de ejemplo a considerar:

Ejemplo #1: Etiquetas cerradas en orden incorrecto

<html><body> Oración aquí** **</cuerpo></html>

En este caso, empujaría todas las etiquetas de apertura a la pila para que se vea así:

<cuerpo>

y, al encontrar una etiqueta de cierre en la cola, querrá verificar esa pila para ver si está presente la coincidencia correcta. La primera etiqueta de cierre que encuentra es ** **; sin embargo, la última etiqueta de apertura (en la parte superior de la pila) es . Eso es malo. Tan pronto como determine que el archivo HTML no es válido,**devolver la pila de etiquetas de apertura**sin quitar la etiqueta de apertura que no coincide. En este caso, el resultado esperado sería una pila que contiene (de abajo hacia arriba): <html><body>

Ejemplo # 2: Clausura etiqueta con No apertura etiqueta
<html><body>Corregir
oración aquí**</ div>**</cuerpo></html>

En este caso, la primera etiqueta de cierre que encuentre (****Nueva romana',serif; color: #222222; background: white;">) coincide con su etiqueta de apertura, pero la siguiente (** **) no lo hace, por lo que el resultado esperado sería una pila que contiene (que va de abajo hacia arriba): **<html><cuerpo>**

Tenga en cuenta que el**
**¡La etiqueta se cierra automáticamente y no debe colocarse en la pila!

Ejemplo #3: Etiqueta de apertura nunca cerrada

<html><cuerpo>Este es un texto

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. El resultado esperado sería una pila que contiene (que va de abajo hacia arriba):

<html><cuerpo><b>

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

<html><body> Hola, mundo! </body></html>** **

Esto es similar al ejemplo n.° 2, excepto que ahora, cuando encuentra la etiqueta de cierre que no tiene una etiqueta de apertura, la pila está vacía ya que todo lo anterior coincide. Sin embargo, devolver una pila vacía significa que el archivo*es*bien formateado! En este caso, sin embargo, debe volver**nulo**para indicar que el archivo no está bien formateado. Piense en cómo puede diferenciar entre cuándo devolver un valor nulo y cuándo devolver una pila vacía.

Por favor, no cambie la firma del*es válida*método (su lista de parámetros, nombre y tipo de valor de retorno). Además, no cree ningún archivo .java adicional para su solución y no modifique*HTMLTag.java*. Si necesita clases adicionales, puede definirlas en*HtmlValidator.java*. Por último, asegúrese de que su*Validador Html*class está en el paquete predeterminado, es decir, no hay una declaración de "paquete" en la parte superior del código fuente.

Consejos útiles

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

- https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

Consulte esta documentación si necesita ayuda para comprender los métodos que están disponibles para usted.

Tenga en cuenta que su *HtmlValidator.isValidHtml* El método solo debe usar métodos en la interfaz Queue, aunque la cola se implemente mediante LinkedList.

Está bien si tu *esValidHtml* El método modifica el contenido de la cola que se pasa como entrada, por ejemplo, eliminando elementos.

Antes de enviar

Por favor, asegúrese de que:

- su *Validador Html* la clase está en el paquete predeterminado, es decir, no hay una declaración de "paquete" en la parte superior del código fuente
- su *Validador Html* la clase se compila y no ha cambiado la firma del *esValidHtml* método
- no ha creado ningún archivo .java adicional y no ha realizado ningún cambio en *HTMLTag.java* (no necesita enviar este archivo o *HtmlReader.java*)

1. Descarga la distribución JUnit en junit-dist.jar

   Siga estos pasos para agregar la biblioteca.

2. https://intellij-support.jetbrains.com/hc/enus/community/posts/360009909039-How-do-I-add-a-build-Path-to-aclass-folder-

3. Descarga las pruebas en **tarea2-pruebas.jar** y agréguelo a la ruta de compilación del proyecto Eclipse como se indicó anteriormente.

4. También descargue los archivos de entrada de prueba de homework2-files.zip Descomprima este archivo en su computadora y copie los siete archivos .html en su proyecto IntelliJ; debería poder arrastrarlos y soltarlos directamente en IntelliJ. Asegúrese de colocarlos en el directorio raíz de su proyecto, como hizo con los dos archivos .jar.

5. Ahora ejecute las pruebas haciendo clic derecho *tarea2-pruebas.jar* en IntelliJ para obtener el menú emergente/contextual y seleccionar "Ejecutar como -->" y luego "Java

> Aplicación." Debería ver las pruebas ejecutadas en la consola y debería indicarle su puntuación para esta tarea, o "¡Buen trabajo!" si su puntuación fuera del 100%.

Alternativamente, si desea ejecutar el autograder desde la línea de comando, coloque los dos archivos .jar y sus archivos .class para esta tarea en el mismo directorio junto con los archivos .html que descargó y ejecute:

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

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

Esto agregará*junit-dist.jar*y*tarea2-pruebas.jar*al classpath y luego ejecute Java con*Homework2Grader*como clase principal.