Estructuras de Datos 2023-1
Práctica 3: Pilas

Pedro Ulises Cervantes González
confundeme@ciencias.unam.mx

Jorge Macías Gómez
jorgeulmomacias@gmail.com

Emmanuel Cruz Hernández
emmanuel_cruzh@ciencias.unam.mx

Fecha límite de entrega: 29 de septiembre de 2022
Hora límite de entrega: 23:59:59 hrs

1. Actividad

1.1. Actividad 1 *(1 punto)*

Implementa la interfaz `TDAStack` vista en clase en un archivo llamado *Stack.java*. Implementa cada uno de los métodos definidos en la interfaz, usando una implementación basada en referencias usando una clase nodo llamada `Node`.

La complejidad en tiempo de tu implementación por cada método debe ser la siguiente:

- push(): O(1)
- pop(): O(1)
- top(): O(1)
- isEmpty(): O(1)
- clear(): O(1)
- show(): O(1)

1.2. Actividad 2

Con ayuda de la clase `Stack` que implementaste en la actividad 1, implementa un programa que permita almacenar cadenas de caracteres en una pila, dada cadena tiene a lo más 255 caracteres.

Cada localidad de memoria de la pila puede contener alguno de los siguientes elementos:

- Un entero que representa la longitud de la cadena que le antecede en la pila.
- Un caracter que forma parte de una de las cadenas que se almacenan en la pila.

Si tenemos la siguiente representación de una pila:

El elemento que se encuentra en el tope es un *10*, que corresponde a la cantidad de caracteres que conforman la cadena próxima almacenada en la pila. El resto de los caracteres almacenados corresponde a un único caracter de una cadena.

Implementa los siguientes métodos

- `pushString(String)` *(1 punto)*

  Esta operación debe almacenar una cadena de longitud k en la pila. De acuerdo a la construcción anterior, se requerirá insertar k caracteres en la pila y luego insertar el entero k para tener referencia de la longitud de dicha cadena. Así, el último elemento insertado en la pila sería el número k.

10
'R'
'e'
'p'
'o'
's'
'i'
'c'
'i'
'o'
'n'

- `popString()` *(1 punto)*

  Esta operación debe devolver la última cadena insertada en la pila; conforme a la construcción anterior. Además, la cadena y su tamaño deben ser eliminados de la pila.

1.3. Actividad 3 *(2 puntos)*

Con ayuda de la clase `Stack` que implementaste en la actividad 1, implementa el método `isHTMLMatched(String)` que verifique si una cadena contiene etiquetas de HTML balanceadas. Regresa *true* si las etiquetas están correctamente balanceadas o *false* en caso contrario.

Decimos que una cadena con etiquetas HTML está balanceada si toda etiqueta que abre, tiene una etiqueta de cierre, así como mantener la modularidad en las etiquetas.

Por ejemplo:

- isHTMLMatched(" $<$ *body* $><$ *h*1 $><$ /*h*1 $><$ *p* $><$ /*p* $><$ *p* $><$ /*p* $><$ *h*1 $><$ /*h*1 $><$ /*body* $>$ ") $\rightarrow$ **true**
- isHTMLMatched(" $<$ *body* $><$ *h*1 $><$ /*h*1 $><$ *p* $><$ /*p* $><$ *p* $><$ /*p* $><$ *h*1 $><$ /*h*1 $><$ /*body* $>$ ") $\rightarrow$ **false**, ya que hay una etiqueta $<$ *p* $>$ que abre, pero nunca cierra.
- isHTMLMatched(" $<$ *body* $><$ /*h*1 $><$ *p* $><$ /*p* $><$ *p* $><$ /*p* $><$ *h*1 $><$ /*h*1 $><$ /*body* $>$ ") $\rightarrow$ **false**, ya que hay una etiqueta $<$ /*h*1 $>$ que cierra, pero nunca abre.
- isHTMLMatched(" $<$ *body* $><$ *h*1 $><$ /*body* $><$ /*h*1 $><$ *p* $><$ /*p* $><$ *p* $><$ /*p* $><$ *h*1 $><$ /*h*1 $>$ ") $\rightarrow$ **false**, ya que hay una etiqueta $<$ /*body* $>$ que no se encuentra en el nivel adecuado para cerrar la etiqueta $<$ *body* $>$.

1.4. Actividad 4 *(3 puntos)*

El problema de las N reinas consiste en colocar N reinas del juego de ajedrez en un tablero de ajedrez de NxN tal que no exista una pareja de reinas que se ataquen mutuamente. Para este ejercicio, utiliza los Arraylists de Java para crear un tablero de un tamaño N dado por el usuario, una vez creado, debes imprimir todas las soluciones posibles.
Debes utilizar backtracking para la resolución del problema.

1.5. Actividad 5 *(3 puntos)*

Dado un Laberinto como una matriz de valores binarios de dimensiones N*N. Donde el origen será la casilla 0,0 y el destino la casilla N-1,N-1. Tal que el valor 0 en la matriz, representa una espacio vació, y el valor 1 representa una pared, como en la figura 1. Donde podemos ver el camino para resolver nuestro laberinto.
Elabora un programa que utilice backtracking para resolver el laberinto.
Hint. ¿Una vez que visitas una casilla, qué deberías hacer?

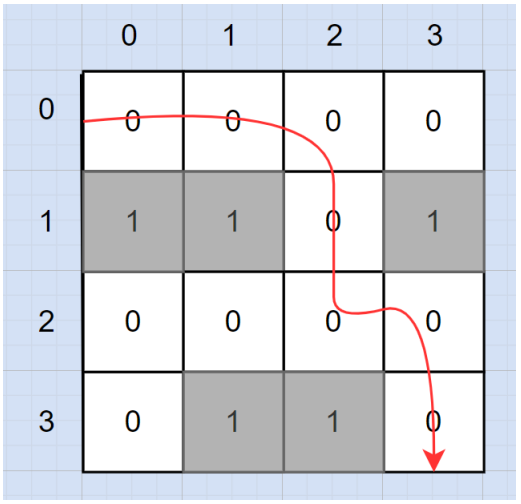

Figura 1: Representación laberinto

2. Reglas Importantes

- No se recibirán prácticas en las que estén involucrados más de dos integrantes.
- Para esta práctica queda prohibido utilizar listas o pilas implementadas por el API de Java.
- Cumple con los lineamientos de entrega.
- Todos los archivos deberán contener nombre y número de cuenta.
- Tu código debe estar comentado. Esto abarca clases, atributos, métodos y comentarios extra que consideres necesarios.
- Utiliza correctamente las convenciones para nombrar variables, constantes, clases y métodos. Debes utilizar saltos de linea y espacios según creas conveniente, tú programa debe ser legible.
- El programa debe ser robusto.
- No se reciben entregas fuera del Classroom
- En caso de no cumplirse alguna de las reglas especificadas, se restará 0.5 puntos en tu calificación obtenida.