



APRENDERAPROGRAMAR.COM

LA ESTRUCTURA DE DATOS
PILA EN JAVA. CLASE STACK
DEL API JAVA. EJEMPLO Y
EJERCICIOS RESUELTOS.
(CU00923C)

Sección: Cursos

Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2039

Resumen: Entrega nº23 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra y José Luis Cuenca

CLASE STACK

A continuación vamos a explicar la clase Stack de Java, que implementa la interface List. Stack se traduce por "pila" y para recordar su significado podemos pensar en una pila de libros. También veremos las características más importantes de esta nueva implementación y haremos un ejemplo a modo de ejercicio.



STACK

La clase Stack es una clase de las llamadas de tipo LIFO (Last In - First Out, o último en entrar - primero en salir). Esta clase hereda de la clase que ya hemos estudiado anteriormente en el curso Vector y con 5 operaciones permite tratar un vector a modo de pila o stack.

Las operaciones básicas son **push** (que introduce un elemento en la pila), **pop** (que saca un elemento de la pila), **peek** (consulta el primer elemento de la cima de la pila), **empty** (que comprueba si la pila está vacía) y **search** (que busca un determinado elemento dentro de la pila y devuelve su posición dentro de ella).

Esta clase es muy sencilla y al crear un objeto de tipo Stack con el constructor básico evidentemente no contendrá ningún elemento.

Un conjunto mucho más completo y consistente para operaciones de stack LIFO es proporcionado en la interface Deque y sus implementaciones, pero nosotros de momento vamos a limitarnos al estudio de la clase Stack.

EJEMPLO USO CLASE STACK

Realizaremos un ejemplo a modo de uso de pila. Uno de los casos más usados en informática de una pila es el de querer verificar si una determinada sentencia o instrucción está equilibrada en cuanto a número de paréntesis, corchetes o llaves de apertura y cierre. Cuando se escribe código de programación si no existe equilibrio entre signos de apertura (por ejemplo un paréntesis de apertura) y cierre (por ejemplo un paréntesis de cierre) ni siquiera debería procesarse la sentencia ya que no estaría formalmente bien construida. De esto se encargan los analizadores léxicos de los compiladores.

Así que vamos a utilizar esta vez tan solo una clase Programa con el método main, donde vamos a ir analizando una sentencia para verificar si es equilibrada o no en símbolos de paréntesis, recorriendo todos sus caracteres desde el inicio hasta el final.

Iremos construyendo nuestra pila apilando un símbolo (cada vez que detectemos un símbolo de apertura o desapilando de ella cuando detectemos un símbolo de cierre. Tendremos que ir analizando

todos los caracteres de una expresión y actuar cuando detectemos un paréntesis, operando en función de si el paréntesis leído es de abrir ("(") o cerrar (")"). El equilibrio en la escritura vendrá determinado al terminar el análisis en función de si la pila está vacía (hay equilibrio) o contiene algún elemento (no hay equilibrio).

Ejemplo: analizamos la expresión "Hay varios países (México, España) que comparten el mismo idioma (español o castellano)."

El resultado al finalizar el análisis de la sentencia sería que la pila está vacía, y esto querrá decir que nuestra sentencia es equilibrada en paréntesis y por tanto el resultado es correcto.

Si analizáramos la expresión "Hay varios países (México, España) que comparten el mismo idioma (español o castellano."

El resultado al finalizar el análisis será que la pila contiene un paréntesis, lo que quiere decir que la expresión no es equilibrada y no tiene el mismo número de paréntesis abiertos que cerrados.

Tendremos que tener en cuenta casos especiales como una expresión cuyo primer elemento sea un paréntesis de cierre. Por ejemplo: "Hay varios países)México, España(" la consideraríamos una expresión incorrecta ya que si la pila está vacía el primer elemento siempre tendrá que ser un paréntesis de apertura y no uno de cierre. Tendremos en cuenta por tanto que además de equilibrio exista corrección en la forma de construcción (que no puedan existir cierres de paréntesis que no se hayan abierto).

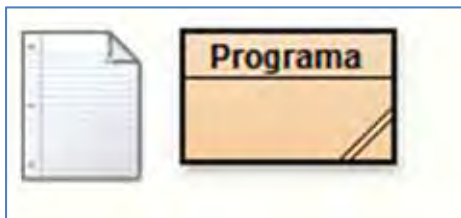
Vamos a escribir ahora el siguiente código con el que vamos a trabajar:

```
/* Ejemplo Interface List, clase Stack aprenderaprogramar.com */
import java.util.Stack;
public class Programa {
    public static void main(String arg[]) {
        String cadenano = "(Cadena no equilibrada en paréntesis(000000))";
        String cadenasi = "(Cadena equilibrada en parentesis())";
        System.out.println("Verificación equilibrado en paréntesis para cadenano:");
        System.out.println(verificaParentesis(cadenano));
        System.out.println("Verificación equilibrado en paréntesis para cadenasi:");
        System.out.println(verificaParentesis(cadenasi));
    }

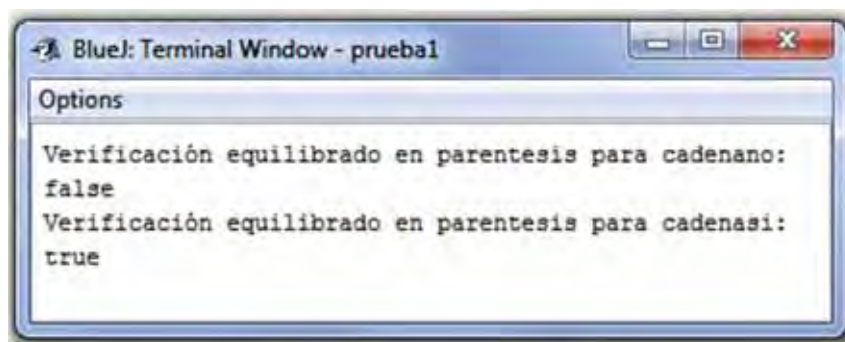
    public static boolean verificaParentesis(String cadena) {
        Stack<String> pila = new Stack<String>();    int i = 0;
        while (i<cadena.length()) { // Recorremos la expresión carácter a carácter
            if(cadena.charAt(i)=='(') {pila.push("(");} // Si el paréntesis es de apertura apilamos siempre
            else if (cadena.charAt(i)=='') { // Si el paréntesis es de cierre actuamos según el caso
                if (!pila.empty()){ pila.pop(); } // Si la pila no está vacía desapilamos
                else { pila.push(""); } break; } // La pila no puede empezar con un cierre, apilamos y salimos
            }
            i++;
        }
        if(pila.empty()){ return true; } else { return false; }
    }
}
```

En este ejemplo hemos creado la función `verificaParentesis` que nos devuelve un boolean indicando si dada una cadena, esta está equilibrada y correcta en paréntesis. Para ello se hace uso internamente en este método de una pila o stack. Así el programa principal `main` tan solo llama a esta función con una cadena de ejemplo (`cadenano` o `cadenasi`) para verificar su equilibrado y corrección en paréntesis.

El diagrama de clases por tanto para BlueJ es muy sencillo y tiene tan solo la clase Programa:



La salida que obtendremos por consola será similar a esta:



```
BlueJ: Terminal Window - prueba1
Options
Verificación equilibrado en parentesis para cadenano:
false
Verificación equilibrado en parentesis para cadenasi:
true
```

CONCLUSIONES

Hemos visto un claro ejemplo del uso de la clase `Stack` que aunque muy sencilla, es muy útil ya que su implementación es muy fácil de aprender con tan solo los 5 métodos comentados anteriormente (`push`, `pop`, `peek`, `empty`, `search`).

EJERCICIO

Crea una clase denominada `VerificadorVocales` con un método capaz de recibir una cadena y una vocal y devolver un resultado que será 1 si el número de la vocal pasada como parámetro es par (hay equilibrio en esa vocal), -1 si el número de la vocal pasada como parámetro es impar (no hay equilibrio en esa vocal) ó 0 (no existe esa vocal en la cadena). Para ello debes utilizar una pila (`Stack`).

Crea una clase con el método `main` donde se pida al usuario una cadena y se le devuelva un mensaje informativo sobre si el números de a's, e's, i's, o's, u's es par o impar. El programa debe dar opción al usuario a seguir introduciendo cadenas si lo desea.

Ejemplo de una posible ejecución del programa:

¿Qué cadena desea analizar?

asgdhfjfgllijgfhdfghjdfjtejfhgfhfgfgejjfjhufjjfmfgffkhfghkfgfkngtblgbu

Resultado: El número de aes es impar, el número de ees es par, el número de ies es impar, el número de oes es 0 y el número de ues es par.

¿Desea analizar otra cadena? N

Gracias por utilizar el programa

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00924C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180