UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ALUMNO:

PAZ MALDONADO CARLOS SAÚL

NOMBRE DEL PROFESOR:

HERNANDEZ CABRERA JESUS

NOMBRE DE LA MATERIA:

ESTRUCTURA DE DATOS

FECHA DE ENTREGA:

25 de septiembre del 2024

TAREA NO. 9

1. Instrucciones de tarea.

Crear la aplicación de la Pila para la revisión del balanceo de paréntesis y llaves.

- Programar el ADT pila(Stack)
- Implementar el programa que, dada una entrada de texto, verifique si este contiene llaves balanceadas es decir que cada llave "{" tiene su respectiva llave de cierre "}". De igual forma con los paréntesis "(" y ")".
- Programar una función o método main con un texto de prueba.

NOTAS:

Para la entrada de texto a evaluar, en lugar de ingresar un String, donde estaría el código a evaluar, se implementa la clase Files y Paths para leer un archivo txt o java, que es donde estará el código a evaluar.

Se puede modificar la url del archivo, en este caso se anexa un archivo txt con nombre “ClaseDePrueba.txt”, en él está el código de prueba. Por ello, se puede remplazar en la clase main la ruta de otro archivo que se requiera verificar.

2. **Capturas de la consola ejecutando el programa.**

Caso 1 (Llaves y paréntesis balanceados)

```
"C:\Program Files\Java\jdk-22.0.2\bin\java.exe" --enable-preview "-javaagent
Está balanceado

Process finished with exit code 0
```

Caso 2 (Borrando intencionalmente una llave y un paréntesis)

```
private static void imp(String str)
    System.out.println(str);
}
```

```
  imp(colaClinica.siguiente().toString);
```

Salida caso 2:

```
"C:\Program Files\Java\jdk-22.0.2\bin\java.exe" --enable-preview "-javaage
No está balanceado

Process finished with exit code 0
```

3. Código

a. Clase Main

```
package unam.fesaragon.estructuradatos;

import unam.fesaragon.estructuradatos.codigoJava.CodigoJava;

public class Main {
    public static void main(String[] args) {
        CodigoJava probando = new
CodigoJava("src/unam/fesaragon/estructuradatos/ClaseDePrueba.txt");
        probando.ejecutarCodigo();
    }
}
```

b. Clase CodigoJava

```
package unam.fesaragon.estructuradatos.codigoJava;

import unam.fesaragon.estructuradatos.ADT.ADTStack;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.IOException;

public class CodigoJava {
    private String codigoJavaEnString;
    private String urlArchivo;
    private ADTStack<Character> pilaLlavesOParentesis;

    public CodigoJava(String urlArchivo) {
        this.urlArchivo = urlArchivo;
        try {
        this.codigoJavaEnString = new
String(Files.readAllBytes(Paths.get(urlArchivo)));
        }catch (IOException e){System.out.println(e);}
        pilaLlavesOParentesis = new ADTStack<>();
    }

    public CodigoJava() {
        pilaLlavesOParentesis = new ADTStack<>();
    }

    public void ejecutarCodigo() {
        //Comprobación de errores con el código cargado
        boolean estaBalanceado = cargarPilaBalanceo(codigoJavaEnString);
        System.out.println((estaBalanceado) ? "Está balanceado": "No está
balanceado");
    }

    private boolean cargarPilaBalanceo(String codigo) {
        boolean estaBalanceado = true;
        for (Character caracter : codigo.toCharArray()) {
            if (caracter == '{' || caracter == '(') {
                pilaLlavesOParentesis.push(caracter);
            } else if (caracter == '}' || caracter == ')') {
                if (pilaLlavesOParentesis.isEmpty()) {
                    estaBalanceado = false;
                    break;
                }
                char caracterApertura = pilaLlavesOParentesis.pop();
                if ((caracter == '}' && caracterApertura != '{') ||
(caracter == ')' && caracterApertura != '(')) {
                    estaBalanceado = false;
                    break;
                }
            }
        }

        return estaBalanceado;
```

```
    }
}
```

4. Clase ADTStack

```
package unam.fesaragon.estructuradatos.ADT;

public class ADTStack<T> {
    private ListaDoblementeLigada<T> datos;

    public ADTStack() {
        datos = new ListaDoblementeLigada<T>();
    }
    public boolean isEmpty(){
        return datos.esta_vacia();
    }
    public int length(){
        return datos.get_tamanio();
    }
    public T pop(){
        T datoASacar = datos.obtener(datos.get_tamanio()-1);
        datos.eliminar_el_final();
        return datoASacar;
    }
    public T peek(){
        return datos.obtener(datos.get_tamanio()-1);
    }
    public void push(T datoAInsertar){
        datos.agregar_al_final(datoAInsertar);
    }

    public void imprimirStack(){
        datos.transversal();
    }
}
```

5. Clase ListaDoblementeLigada

```
package unam.fesaragon.estructuradatos.ADT;

public class ListaDoblementeLigada<T> {
    private NodoDoble<T> head;
    private NodoDoble<T> tail;
    private int tamanio;

    // Constructor
    public ListaDoblementeLigada() {
        this.head = null;
        this.tail = null;
        this.tamanio = 0;
    }

    public ListaDoblementeLigada(NodoDoble<T> head, NodoDoble<T> tail) {
        this.head = head;
        this.tail = tail;
    }

    // Comprobar si está vacía
    public boolean esta_vacia() {
        return this.tamanio == 0;
    }

    // Agregar al inicio de la lista
    public void agregar_al_inicio(T valor) {
```

```
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (esta_vacia()) {
            this.head = nuevo;
            this.tail = nuevo;
        } else {
            nuevo.setSiguiente(this.head);
            this.head.setAnterior(nuevo);
            this.head = nuevo;
        }
        tamanio++;
    }

    // Agregar al final de la lista
    public void agregar_al_final(T valor) {
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (esta_vacia()) {
            this.head = nuevo;
            this.tail = nuevo;
        } else {
            this.tail.setSiguiente(nuevo);
            nuevo.setAnterior(this.tail);
            this.tail = nuevo;
        }
        tamanio++;
    }

    // Agregar después de un nodo de referencia
    public void agregar_después_de(T referencia, T valor) {
        if (esta_vacia()) {
            System.out.println("La lista esta vacia y por lo tanto no
existe la referencia");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(referencia)) {
                break;
            }
            aux = aux.getSiguiente();
        }

        if (aux == null) {
            System.out.println("El nodo de referencia no existe");
            return;
        }

        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        nuevo.setSiguiente(aux.getSiguiente());
        nuevo.setAnterior(aux);
        //Nodo de referencia, no es la ultima en la lista.
        if (aux.getSiguiente() != null) {
            aux.getSiguiente().setAnterior(nuevo);
        } else {
            this.tail = nuevo;
        }
        aux.setSiguiente(nuevo);
        this.tamanio++;
    }

    // Obtener el elemento en una posición específica
    public T obtener(int posicion) {
        //Condiciones
        if (esta_vacia()) {
            return null;
        }
```

```
        if (posicion > this.tamanio || posicion<0) {
            System.out.println("Posición fuera de rango");
            return null;
        }

        NodoDoble<T> aux = this.head;
        for (int i = 0; i < posicion; i++) {
            aux = aux.getSiguiente();
        }

        return aux.getDato();
    }

    // Eliminar el primer elemento
    public void eliminar_el_primero() {
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return;
        }
        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.head = this.head.getSiguiente();
            this.head.setAnterior(null);
        }
        tamanio--;
    }

    // Eliminar el último elemento
    public void eliminar_el_final() {
        if (esta_vacia()) {
            return;
        }

        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.tail = this.tail.getAnterior();
            this.tail.setSiguiente(null);
        }
        tamanio--;
    }

    // Eliminar un elemento en una posición específica
    public void eliminar(int posicion) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }
        if (posicion > this.tamanio) {
            System.out.println("Posición fuera de rango");
            return;
        }

        if (posicion == 1) {
            eliminar_el_primero();
            return;
        }
        if (posicion == this.tamanio) {
            eliminar_el_final();
            return;
        }

        NodoDoble<T> aux = this.head;
```

```
        int i = 1;
        while (i < posicion) {
            aux = aux.getSiguiente();
            i++;
        }

        aux.getAnterior().setSiguiente(aux.getSiguiente());
        aux.getSiguiente().setAnterior(aux.getAnterior());
        this.tamanio--;
    }

    // Buscar un elemento y retornar su posición
    public int buscar(T valor) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return 0;
        }

        NodoDoble<T> aux = this.head;
        int posicion = 1;
        while (aux != null) {
            if (aux.getDato().equals(valor)) {
                return posicion;
            }
            aux = aux.getSiguiente();
            posicion++;
        }
        System.out.println("No se encontro el elemento");
        return 0;
    }

    // Actualizar un elemento
    public void actualizar(T a_buscar, T valor) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(a_buscar)) {
                aux.setDato(valor);
                return;
            }
            aux = aux.getSiguiente();
        }
        System.out.println("El elemento valor o elemento no esta en la
lista");
    }

    // Recorrido transversal en una dirección específica
    // Verdadero es valor por defecto y con el falso es de derecha a
izquierda
    public void transversal(boolean porDefecto) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        if (porDefecto) {
            NodoDoble<T> aux = this.head;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getSiguiente();
            }
        } else {
```

```
            NodoDoble<T> aux = this.tail;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getAnterior();
            }
        }
        System.out.println("NULL");
    }
    //Sobrecargue el metodo para poner por default el recorrido de
izquierda a derecha
    public void transversal() {
        transversal(true);
    }

    //GETTERS Y SETTERS
    // Obtener el tamaño de la lista
    public int get_tamanio() {
        return this.tamanio;
    }
}
```

6. Clase NodoDoble

```
package unam.fesaragon.estructuradatos.ADT;

public class NodoDoble<T> {
    private T dato;
    private NodoDoble<T> anterior;
    private NodoDoble<T> siguiente;

    // Constructores
    public NodoDoble() {
    }

    public NodoDoble(T dato) {
        this.dato = dato;
        this.anterior = null;
        this.siguiente = null;
    }

    public NodoDoble(T dato, NodoDoble<T> anterior, NodoDoble<T>
siguiente) {
        this.dato = dato;
        this.anterior = anterior;
        this.siguiente = siguiente;
    }

    // Getters y Setters
    public T getDato() {
        return dato;
    }

    public void setDato(T dato) {
        this.dato = dato;
    }

    public NodoDoble<T> getAnterior() {
        return anterior;
    }

    public void setAnterior(NodoDoble<T> anterior) {
        this.anterior = anterior;
    }

    public NodoDoble<T> getSiguiente() {
```

```
        return siguiente;
    }

    public void setSiguiente(NodoDoble<T> siguiente) {
        this.siguiente = siguiente;
    }
}
```

7. Archivo de prueba ClasePrueba.txt

```
package unam.fesaragon.estructuradatos;


public class ClaseDePrueba {

    public static void main(String[] args) {
        ColaADT<Paciente> colaClinica = new ColaADT<>();
        imp("\n\nAgregando 3 pacientes a la cola");
        colaClinica.encolar(new Paciente("Alexis", "Martinez Prado", 19,
"Hombre"));
        colaClinica.encolar(new Paciente("Alejandra", "Heredia Nava", 24,
"Mujer"));
        colaClinica.encolar(new Paciente("Luis Angel", "Cervantes Moreno",
29, "Hombre"));
        imp("Mostrando contenido de la cola");
        imp(colaClinica.toString());
        imp("\nEl paciente que sigue es (Sin sacar de la cola): ");
        imp(colaClinica.siguiente().toString());
        imp("Comprobando que no se saco de la cola");
        imp(colaClinica.toString());
        imp("\nAtendiendo al siguiente: " + colaClinica.desEncolar());
        imp("\nMostrando el contenido de la cola nuevamente: ");
        imp(colaClinica.toString());
        imp("\nAgregando dos pacientes nuevos a la cola: ");
        colaClinica.encolar(new Paciente("Leslie", "Lopez Hernandez", 14,
"Mujer"));
        colaClinica.encolar(new Paciente("Ana", "Nieves Clieto", 20,
"Mujer"));
        imp(colaClinica.toString());

    }

    private static void imp(String str) {
        System.out.println(str);
    }
}
```