

ALMACENAMIENTO DE NODOS

Este proyecto de Java proporciona una implementación básica de un sistema de almacenamiento de nodos que se comunican entre sí a través de sockets TCP. El sistema incluye nodos con roles de líder y seguidor, así como un manejador de roles para asignar líderes y realizar operaciones de lectura y escritura.

CLASES

Nodo

La clase **Nodo** representa un nodo en el sistema. Cada nodo puede tener un rol de líder o seguidor. Si un nodo es un seguidor, inicia un servidor TCP para aceptar conexiones entrantes de otros nodos. Los nodos se pueden unir como observadores para recibir mensajes. Además, un nodo puede escribir contenido y notificar a otros nodos sobre la escritura.

```
class Nodo implements Observador {
    private String nombre;
    private Rol rol;
    private List<Observador> observadores = new ArrayList<>();
    private List<String> contenidoEscrito = new ArrayList<>();
    private ServerSocket serverSocket;

    public Nodo(String nombre, Rol rol) { ...

    private void iniciarServidor() { ...

    private void manejarConexionEntrante(Socket clientSocket) { ...

    private void enviarMensaje(String mensaje, int puertoDestino) { ...

    public void unirseComoObservador(Observador observador) { ...

    @Override
    public void recibirMensaje(String mensaje) { ...

    public String getNombre() { ...

    public Rol getRol() { ...

    public void escribirContenido(String contenido, int puertoDestino) { ...

    public List<String> obtenerContenidoEscrito() { ...
}
```

ManejadorRoles

La clase **ManejadorRoles** gestiona la asignación de líderes y realiza operaciones de lectura y escritura en el sistema. Permite agregar nodos al sistema, asignar líderes y realizar operaciones de lectura y escritura.

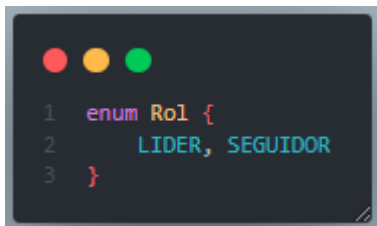
```

class ManejadorRoles {
    private List<Nodo> nodos = new ArrayList<>();

    public void agregarNodo(Nodo nodo) { ...
    public void asignarLider(Nodo nodo) { ...
    public void realizarLectura(Nodo nodoLector) { ...
    public void realizarEscritura(Nodo nodo, String contenido) { ...
    private void enviarMensajeAEscritores(String mensaje) { ...
    public Nodo obtenerNodoPorNombre(String nombre) { ...
    private void mostrarContenidoEscrito(Nodo lider) { ...
}

```

ROLES



```

1  enum Rol {
2      LIDER, SEGUIDOR
3  }

```

Líder

Un nodo con el rol de líder puede escribir contenido y asignar líderes.

Seguidor

Un nodo con el rol de seguidor inicia un servidor para aceptar conexiones y puede recibir mensajes del líder.

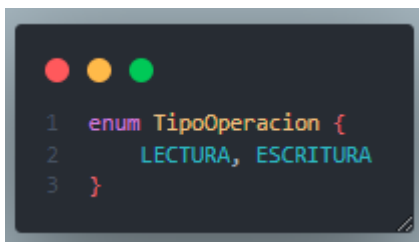
OPERACIONES

Realizar Lectura

Un nodo realiza una operación de lectura para ver el contenido escrito por todos los nodos.

Realizar Escritura

Solo el líder puede realizar operaciones de escritura. Cuando lo hace, notifica a los seguidores sobre la escritura.



```

1  enum TipoOperacion {
2      LECTURA, ESCRITURA
3  }

```

Comunicación en Red

Los nodos se comunican entre sí a través de sockets TCP. Cada nodo actúa como un servidor que acepta conexiones entrantes y como un cliente que se conecta a otros nodos.

```
1 private void iniciarServidor() {
2     try {
3         serverSocket = new ServerSocket(0);
4         System.out.println(nombre + " está escuchando en el puerto " + se
5 rverSocket.getLocalPort());
6
7         // Hilo para aceptar conexiones entrantes
8         new Thread(() -> {
9             while (true) {
10                 try {
11                     Socket clientSocket = serverSocket.accept();
12                     new Thread(() -> manejarConexionEntrante(clientSocket)).start();
13                 } catch (IOException e) {
14                     e.printStackTrace();
15                 }
16             }
17         }).start();
18     } catch (IOException e) {
19         e.printStackTrace();
20     }
21
22     private void manejarConexionEntrante(Socket clientSocket) {
23         try (BufferedReader reader = new BufferedReader(new InputStreamReader
24 (clientSocket.getInputStream()))) {
25             String mensaje = reader.readLine();
26             recibirMensaje(mensaje);
27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30     }
31 }
```

USO

El programa proporciona un menú interactivo para realizar las siguientes operaciones:

1. **Agregar Nodo:** Agrega un nuevo nodo al sistema.
2. **Realizar Lectura:** Realiza una operación de lectura para ver el contenido escrito por todos los nodos.
3. **Realizar Escritura:** Realiza una operación de escritura si el nodo es el líder.
4. **Salir:** Sale del programa.

```
=== Menú ===  
1. Establecer conexión a SERVIDOR  
2. Agregar Nodo  
3. Realizar Lectura  
4. Realizar Escritura  
5. Salir  
Ingrese la opción:
```

Consideraciones

- La comunicación en red está implementada utilizando sockets TCP.

CLIENTE DE ALMACENAMIENTO

Este programa Java representa el cliente de almacenamiento que permite interactuar con un sistema de almacenamiento de nodos. El sistema incluye nodos con roles de líder y seguidor que se comunican entre sí mediante sockets TCP. A continuación, se describen las características y la funcionalidad del programa:

MENÚ DE OPCIONES

1. **Establecer conexión a SERVIDOR:** Intenta establecer una conexión con el servidor del sistema de almacenamiento.
2. **Agregar Nodo:** Agrega un nuevo nodo al sistema de almacenamiento.
3. **Realizar Lectura:** Realiza una operación de lectura para ver el contenido escrito por todos los nodos.
4. **Realizar Escritura:** Realiza una operación de escritura si el nodo es el líder.
5. **Salir:** Sale del programa.

```

1 while (true) {
2     System.out.println("\n=== Menú ===");
3     System.out.println("1. Establecer conexión a SERVIDOR");
4     System.out.println("2. Agregar Nodo");
5     System.out.println("3. Realizar Lectura");
6     System.out.println("4. Realizar Escritura");
7     System.out.println("5. Salir");
8
9     try {
10        System.out.print("Ingrese la opción: ");
11        int opcion = scanner.nextInt();
12
13        switch (opcion) {
14            case 1:
15                estaConectado = establecerConexion();
16                break;
17            case 2:
18                if (estaConectado) {
19                    agregarNodo(manejadorRoles, scanner);
20                } else {
21                    System.out.println("No estás conectado a TCPServer. Establece la conexión primero.");
22                }
23                break;
24            case 3:
25                if (estaConectado) {
26                    realizarOperacion(manejadorRoles, TipoOperacion.LECTURA, scanner);
27                } else {
28                    System.out.println("No estás conectado a TCPServer. Establece la conexión primero.");
29                }
30                break;
31            case 4:
32                if (estaConectado) {
33                    realizarOperacionEscritura(manejadorRoles, scanner);
34                } else {
35                    System.out.println("No estás conectado a TCPServer. Establece la conexión primero.");
36                }
37                break;
38            case 5:
39                System.out.println("Saliendo del programa.");
40                System.exit(0);
41            default:
42                System.out.println("Opción no válida. Por favor, ingrese una opción válida.");
43        }
44    } catch (java.util.InputMismatchException e) {
45        System.out.println("Entrada no válida. Por favor, ingrese un número entero.");
46        scanner.nextLine(); // Limpiar el buffer del scanner
47    }
48 }

```

FUNCIONALIDADES

- **Establecer Conexión a SERVIDOR:** Intenta establecer una conexión con el servidor en el puerto 9999 de localhost. Si la conexión es exitosa, muestra un mensaje indicando que la conexión se ha establecido. Si hay un error, imprime un mensaje de error y regresa **false**.

```

private static boolean establecerConexion() {
    try (Socket client = new Socket(InetAddress.getByAddress(host:"localhost"), port:9999)) {
        System.out.println(x:"Conexión establecida a SERVIDOR.");
        return true;
    } catch (Throwable t) {
        System.out.println(x:"ERROR. El SERVIDOR esta apagado encender por favor");
        return false;
    }
}

```

- **Agregar Nodo:** Permite al usuario ingresar el nombre y el rol del nodo (líder o seguidor) y agrega un nuevo nodo al sistema de almacenamiento utilizando el

ManejadorRoles.

```
private static void agregarNodo(ManejadorRoles manejadorRoles, Scanner scanner) {
    System.out.print(s:"Ingrese el nombre del nodo: ");
    String nombre = scanner.next();

    System.out.println(x:"Seleccione el rol del nodo:");
    System.out.println(x:"1. Líder");
    System.out.println(x:"2. Seguidor");
    int opcionRol = scanner.nextInt();

    Rol rol = (opcionRol == 1) ? Rol.LIDER : Rol.SEGUIDOR;

    Nodo nodo = new Nodo(nombre, rol);
    manejadorRoles.agregarNodo(nodo);
}
```

- **Realizar Lectura:** Permite al usuario ingresar el nombre del nodo que realizará la operación de lectura. La operación de lectura muestra el contenido escrito por todos los nodos en el sistema utilizando el **ManejadorRoles**.

```
private static void realizarOperacion(ManejadorRoles manejadorRoles, TipoOperacion tipoOperacion, Scanner scanner) {
    System.out.print(s:"Ingrese el nombre del nodo que realizará la operación: ");
    String nombre = scanner.next();

    Nodo nodo = manejadorRoles.obtenerNodoPorNombre(nombre);

    if (nodo != null) {
        if (tipoOperacion == TipoOperacion.LECTURA) {
            manejadorRoles.realizarLectura(nodo);
        }
    } else {
        System.out.println(x:"Nodo no encontrado. Asegúrese de que el nodo exista.");
    }
}
```

- **Realizar Escritura:** Permite al usuario ingresar el nombre del nodo que realizará la operación de escritura y el contenido a escribir. Solo el líder puede realizar operaciones de escritura, y se notifica a los seguidores sobre la escritura.

```
private static void realizarOperacionEscritura(ManejadorRoles manejadorRoles, Scanner scanner) {
    System.out.print(s:"Ingrese el nombre del nodo que realizará la operación de escritura: ");
    String nombre = scanner.next();

    Nodo nodo = manejadorRoles.obtenerNodoPorNombre(nombre);

    if (nodo != null) {
        if (nodo.getRol() == Rol.LIDER) {
            System.out.print(s:"Ingrese el contenido a escribir: ");
            String contenido = scanner.next();
            manejadorRoles.realizarEscritura(nodo, contenido);
        } else {
            System.out.println(nombre + " no puede realizar una escritura, ya que no es el líder.");
        }
    } else {
        System.out.println(x:"Nodo no encontrado. Asegúrese de que el nodo exista.");
    }
}
```

NOTAS ADICIONALES

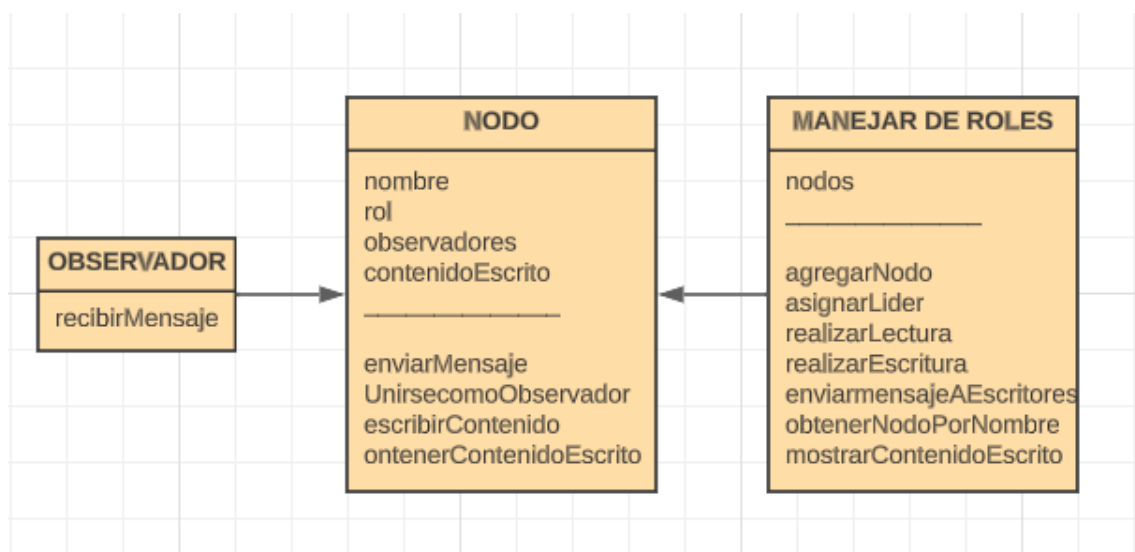
- Si se intenta realizar alguna operación sin haber establecido una conexión con el servidor primero, se muestra un mensaje indicando que es necesario establecer la conexión antes de realizar esa operación.

- Si se ingresa una opción no válida en el menú, se muestra un mensaje indicando que la opción no es válida y se solicita al usuario que ingrese una opción válida.
- Se manejan las excepciones relacionadas con la entrada del usuario para evitar fallos inesperados.
- Se proporciona un mensaje de error específico si no se puede establecer la conexión con el servidor.
- El código está estructurado en métodos para facilitar la lectura y el mantenimiento del programa.

Este programa es un componente del sistema de almacenamiento y debe usarse junto con el servidor y otras clases del sistema para un funcionamiento completo.

ESQUEMA DE ALMACENAMIENTO

1. **Clase Nodo:** Representa un nodo en el sistema. Cada nodo tiene un nombre, un rol (ya sea líder o seguidor), una lista de observadores, y una lista de contenido escrito.
2. **Interfaz Observador:** Define el método **recibirMensaje(String mensaje)**, que permite que los nodos se notifiquen entre sí.
3. **Clase ManejadorRoles:** Gestiona la lógica para agregar nodos, asignar líderes, realizar lecturas y escrituras, y mantener una lista de nodos en el sistema.
4. **Enums Rol y TipoOperacion:** Definen los roles de los nodos (Líder o Seguidor) y los tipos de operaciones (Lectura o Escritura), respectivamente.



LINK GITHUB: <https://github.com/oskr2023/MODULO-ALMACENAMIENTO/blob/main/almacenamiento/Main.java>