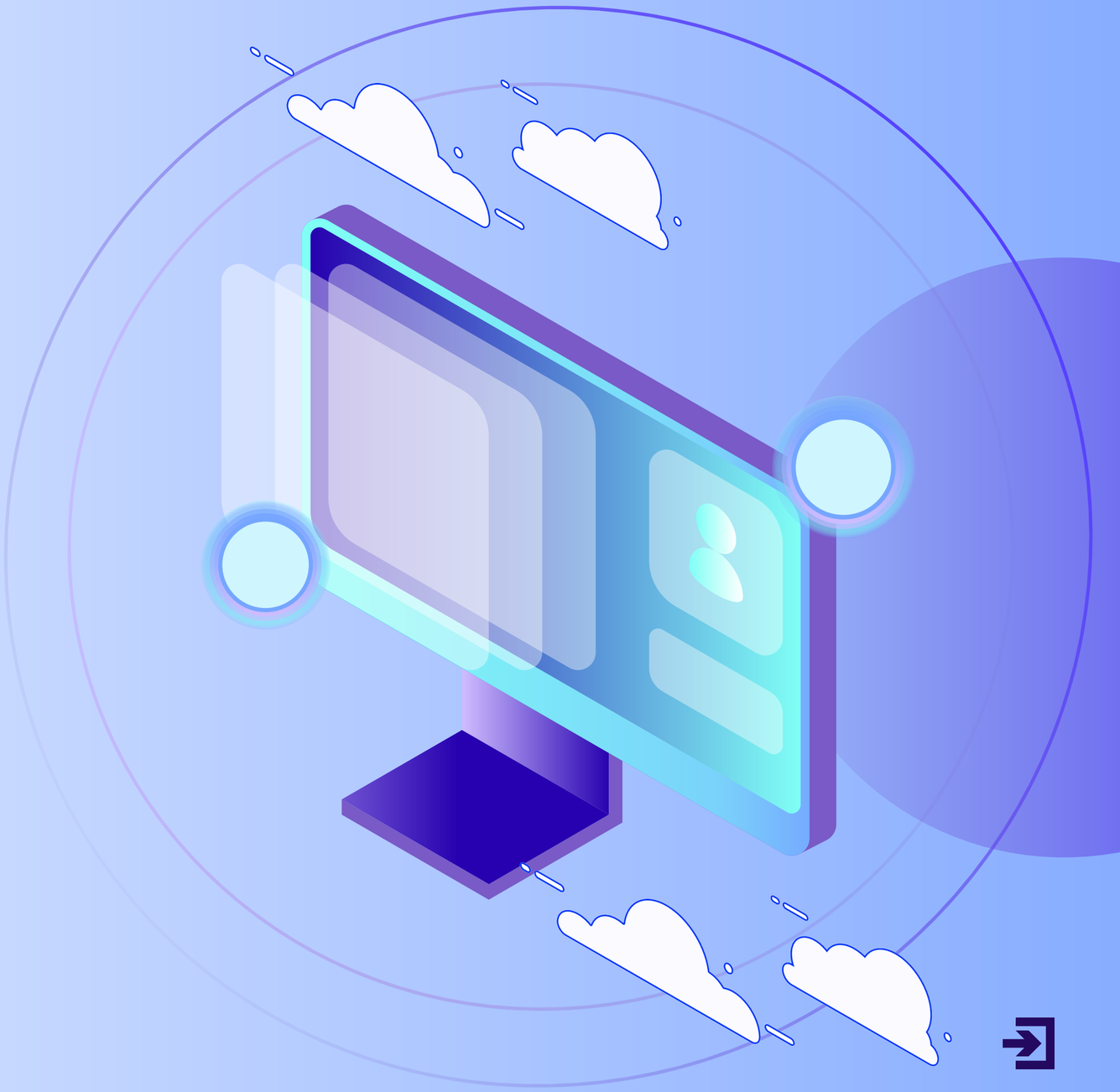
Liceria Tech

TOKEN RING

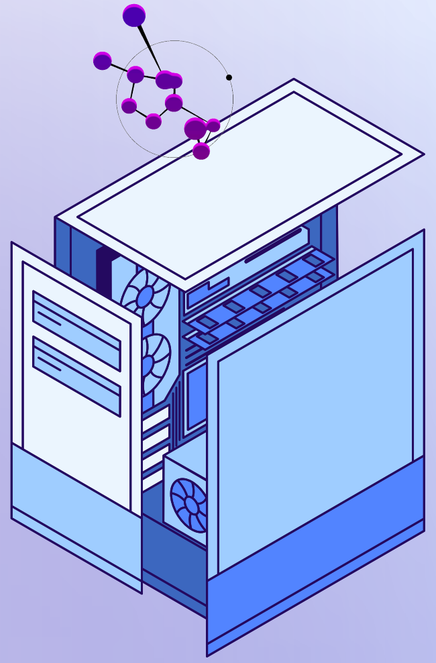


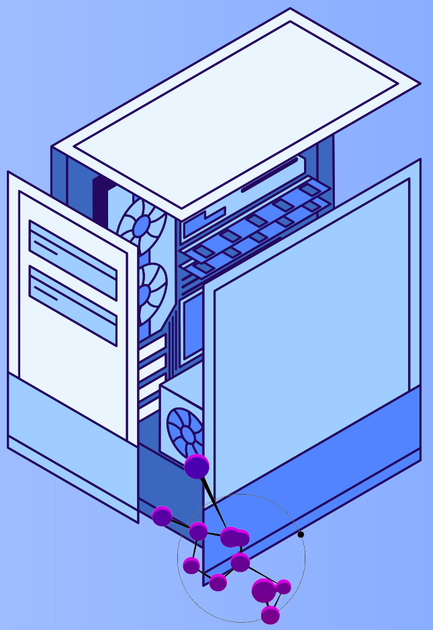
Alejandro Sánchez Quesada

Alejandro Jiménez Sánchez

Liceria Tech

¿QUÉ ES UN TOKEN RING?





Definición:
"Una red Token Ring es un protocolo de comunicación en el que los dispositivos están organizados en forma de anillo. El acceso al medio se controla mediante un TOKEN, un pequeño paquete de datos que circula por la red."

ESTRUCTURA

CLASE MIEMBRO TOKEN

```java
public MiembroToken(int id, int puerto, boolean tieneToken, boolean esUltimo) {
    this.id = id;
    this.puerto = puerto;
    this.tieneToken = tieneToken;
    this.esUltimo = esUltimo;
    this.puertoSiguiente = esUltimo ? 10000 : puerto + 1;
    this.tiempoEspera = id * 1000;
    this.ultimaRecepcion = LocalDateTime.now();
}
```

- id: Identificador del miembro y define su tiempo de espera (1s, 2s, 3s...).
- puerto: Puerto en el que escucha el miembro para recibir el token.
- tieneToken: Indica si el miembro comienza con el TOKEN.
- esUltimo: Si es el último nodo, envía el token de vuelta al primer miembro.
- puertoSiguiente: Determina el puerto del siguiente miembro en la red.

ESTRUCTURA

CONSTRUCTOR

```
public MiembroToken(int id, int puerto, boolean tieneToken, boolean esUltimo) {
    this.id = id;
    this.puerto = puerto;
    this.tieneToken = tieneToken;
    this.esUltimo = esUltimo;
    this.puertoSiguiente = esUltimo ? 10000 : puerto + 1;
    this.tiempoEspera = id * 1000;
    this.ultimaRecepcion = LocalDateTime.now();
}
```

- Inicializa los atributos de la clase MiembroToken.
- Si el miembro es el último de la red (esUltimo = true), el TOKEN volverá al puerto 10000.
- El tiempo Espera se calcula en milisegundos, es proporcional al idmiembro (1s, 2s, 3s, etc.).

ESTRUCTURA

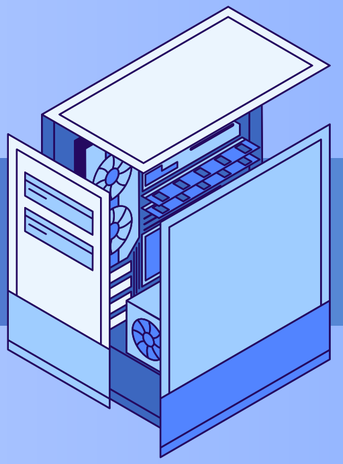
MÉTODO RUN ( )

```java
@Override
public void run() {
    try (DatagramSocket socket = new DatagramSocket(puerto)) {
        System.out.println("Miembro " + id + " escuchando en el puerto " + puerto);

        if (tieneToken) {
            procesarToken();
        }

        byte[] buffer = new byte[256];
        while (true) {
            DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);
            socket.receive(paquete);

            String mensaje = new String(paquete.getData(), offset: 0, paquete.getLength(), StandardCharsets.UTF_8);
            if (mensaje.equals("TRUE")) {
                procesarToken();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

- Se abre un socket UDP para escuchar en el puerto del miembro.
- Si el miembro comienza con el TOKEN, llama a procesarToken().
- Luego entra en un bucle infinito esperando recibir el TOKEN.
- Cuando recibe el mensaje "TRUE", significa que recibió el TOKEN.

ESTRUCTURA

MÉTODO PROCESAR TOKEN ( )



```
private void procesarToken() {  2 usages
    LocalDateTime ahora = LocalDateTime.now();
    if (ultimaRecepcion != null) {
        Duration duracion = Duration.between(ultimaRecepcion, ahora);
        System.out.println("Miembro " + id +
                " recibió el TOKEN. Tiempo desde la última recepción: "
                + duracion.toMillis() + " ms");
    }
    ultimaRecepcion = ahora;

    try {
        System.out.println("Miembro " + id + " está procesando durante "
                + (tiempoEspera / 1000) + " segundos.");
        Thread.sleep(tiempoEspera);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }

    enviarToken();
}
```

- Calcula el tiempo desde la última vez que recibió el TOKEN .
- Simula el uso del recurso asignándole un tiempo de espera (1, 2 o 3 segundos) según el identificador del miembro.
- Después de procesar, envía el TOKEN al siguiente miembro llamando a enviarToken().

ESTRUCTURA

MÉTODO ENVIAR TOKEN ( )

```
private void enviarToken() {  1 usage
    try (DatagramSocket socket = new DatagramSocket()) {
        InetAddress direccion = InetAddress.getByName( host: "localhost");
        byte[] buffer = "TRUE".getBytes(StandardCharsets.UTF_8);
        DatagramPacket paquete = new DatagramPacket(buffer,
                buffer.length, direccion, puertoSiguiente);

        socket.send(paquete);
        System.out.println("Miembro " + id + " envió el TOKEN al puerto "
                + puertoSiguiente);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- Crea un nuevo socket UDP para enviar el TOKEN.
- El TOKEN se representa como un simple mensaje "TRUE".
- Se envía al siguiente puerto (puertoSiguiente), o al primer miembro si es el último nodo.

ESTRUCTURA

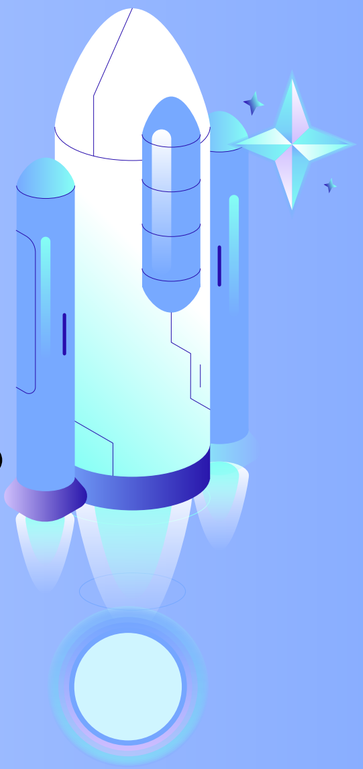
MÉTODO MAIN ( )

```
public static void main(String[] args) {
    if (args.length != 4) {
        System.out.println
                ("Uso: java MiembroToken [id] [puerto] [token_al_inicio] [soy_el_ultimo]");
        return;
    }

    int id = Integer.parseInt(args[0]);
    int puerto = Integer.parseInt(args[1]);
    boolean tieneToken = args[2].equalsIgnoreCase( anotherString: "yes");
    boolean esUltimo = args[3].equalsIgnoreCase( anotherString: "yes");

    MiembroToken miembro = new MiembroToken(id, puerto, tieneToken, esUltimo);
    new Thread(miembro).start();
}
```

- Toma los parámetros de entrada: ID, puerto, si empieza con el TOKEN, y si es el último miembro.
- Crea una instancia de MiembroToken y la ejecuta en un nuevo hilo (Thread), permitiendo que varios miembros corran simultáneamente.

CONCLUSIÓN

"Hemos trabajado en equipo y visto cómo simular una Red Token Ring en Java usando sockets UDP, gestionando la recepción, procesamiento y envío del TOKEN."



GRACIAS POR SU ATENCIÓN!