Token Ring

PRÁCTICA 4

Andres Rodarte López
ESCUELA SUPERIOR DE CÓMPUTO | INSTITUTO POLITÉCNICO NACIONAL

Introducción

Desarrollar un programa en Java, el cual implementará un token que pasará de un nodo a otro nodo, en una topología de anillo. El anillo constará de cuatro nodos, del 0 al 3. El token será un número entero de 8 bytes. El nodo 0 inicializará el token con cero. El nodo 0 enviará el token al nodo 1, entonces el nodo 1 incrementará el token y lo enviará al nodo 2. El nodo 2 incrementará el token y lo enviará al nodo 3. El nodo 3 incrementará el token y lo enviará al nodo 0. Cada nodo deberá desplegar el valor del token cada vez que lo recibe.

Código

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * Node
 */
public class Node {
    private Object lock = new Object();
    private boolean send = false;
    private TokenServer TS;
    private TokenClient TC;
    private long token = 0;
    private int node;

    Node(final int node) {
        this.node = node;
        send = (node == 0)? true:false;
        this.TS = new TokenServer(this.node);
        this.TC = new TokenClient(this.node);
    }

    public void __start(){
        TS.start();
        TC.start();
    }

    /**
     * InnerNode
```

```
     */
    public class TokenServer extends Thread {
        private DataInputStream is;
        private int node;
        private Socket nodeClient;
        public TokenServer(final int node) {
            this.node = node;
        }

        @Override
        public void run() {
            super.run();
            try (ServerSocket serverSocket = new ServerSocket(50000 + this.n
ode)) {
                System.out.println("Starting node: " + this.node);
                nodeClient = serverSocket.accept();
                is = new DataInputStream(nodeClient.getInputStream());

                while (true) {
                    try {
                        token = is.readLong();
                        System.out.println("Recibed token: " + token);
                        synchronized (lock) {
                            send = true;
                            try {
                                sleep(500);
                            } catch (final InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    } catch (Exception e) {
                        System.err.println(e);
                    }
                }
            } catch (final Exception e) {
                System.out.println(e);
            }
        }
    }

    public class TokenClient extends Thread {
        private Socket conexion2server = null;
        private DataOutputStream os;
        private int node;
```

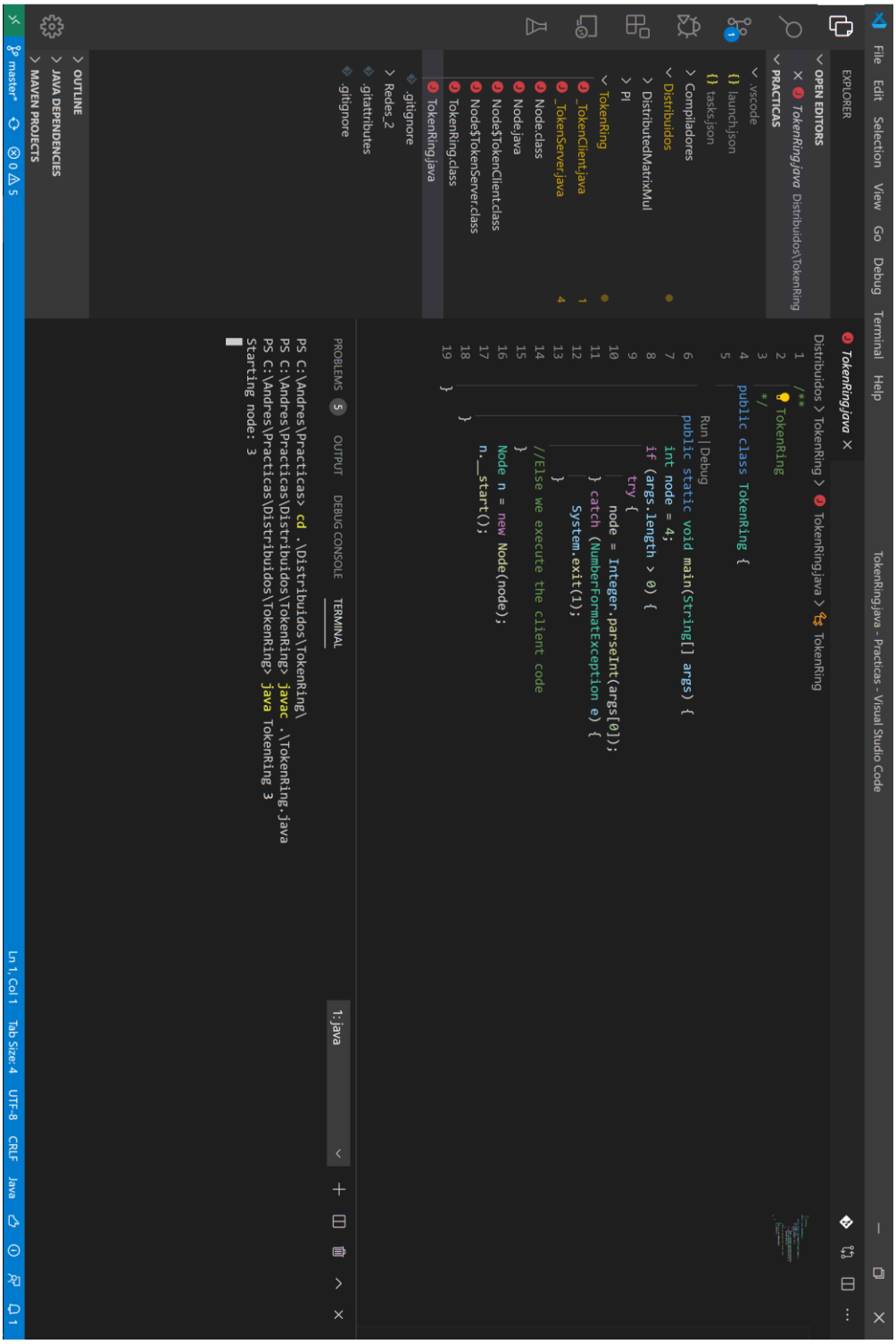
```
        public TokenClient(final int node){
            this.node = node;
        }
        @Override
        public void run() {
            super.run();
            while (true){
                try {
                    this.conexion2server = new Socket("localhost", 50000 + (
this.node + 1) % 4);
                    this.os = new DataOutputStream(this.conexion2server.getO
utputStream());
                    System.out.println("Conected to server:" + (50000 + (thi
s.node + 1) % 4));
                    break;
                } catch (final Exception e) {
                    try {
                        sleep(100);
                    } catch (InterruptedException e1) {
                        e1.printStackTrace();
                    }
                }
            }
            while(true){
                try {
                    synchronized (lock) {
                        if(send){
                            os.writeLong(++token);
                            send = false;
                        }
                        try {
                            sleep(10);
                        } catch (final InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                } catch (Exception e) {
                    System.err.println(e);
                }
            }
        }
    }

}
```

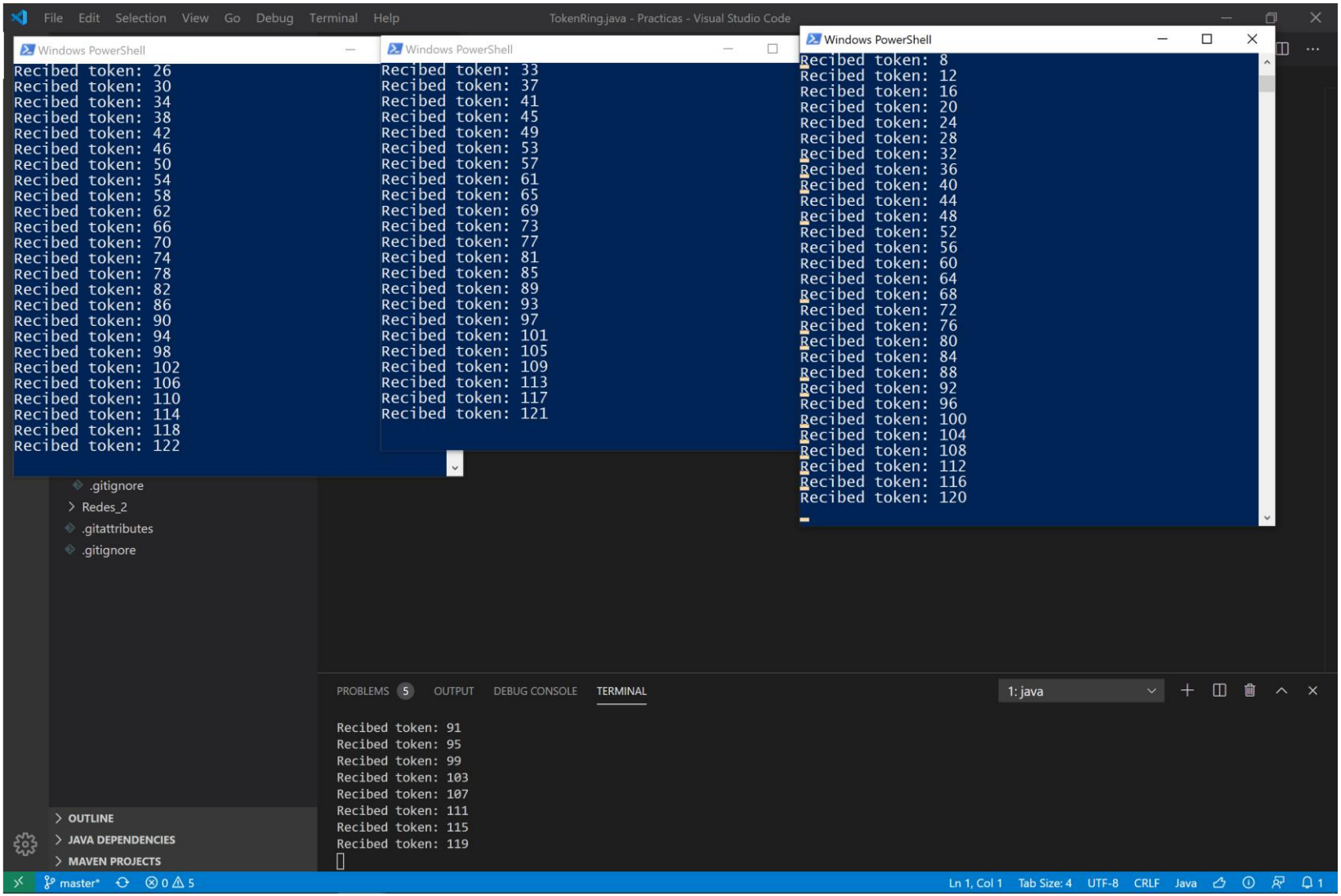
Resumen

El algoritmo funciona mediante dos threads. Uno se encarga de controlar el socket de entrada como cliente y el otro el socket de salida, como servidor. Se conecta al nodo siguiente numerado. Y recibe del anterior en token ring. Se usa un objeto lock, el cual permite sincronizar ambos relojes.

Capturas



La imagen muestra la clase principal del proyecto. Además de su compilación y su ejecución con numero de nodo3.



Ejecución de los cuatro nodos.