

A dark green vertical bar is positioned on the left side of the slide. A green arrow-shaped banner points to the right from this bar, containing the date. Below the banner, several thin, curved lines in dark green and light gray extend upwards from the bottom left corner.

10-9-2021

# Token-Ring

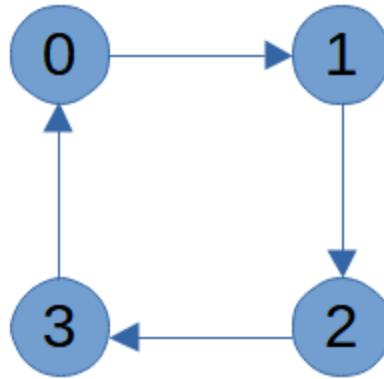
Martínez Coronel Brayan Yosafat

Desarrollo de Sistemas Distribuidos

PINEDA GUERRERO CARLOS

## Objetivo

Desarrollar un programa en Java, el cual implementará un token que se enviará de un nodo a otro nodo mediante **sockets seguros**, en una topología lógica de anillo. El anillo constará de cuatro nodos:



El token será un número entero de 64 bits.

1. Al principio el nodo 0 enviará el token al nodo 1.
2. El nodo 1 recibirá el token y lo enviará al nodo 2.
3. El nodo 2 recibirá el token y lo enviará al nodo 3.
4. El nodo 3 recibirá el token y lo enviará al nodo 0.
5. El nodo 0 recibirá el token y lo enviará al nodo 1.
6. Ir al paso 2.

Cuando un nodo reciba el token lo incrementará en 1 y desplegará el valor del token.

Cuando la cuenta en el nodo 0 llegue a 1000, el nodo 0 deberá terminar su ejecución.

Vamos a probar el programa en una sola computadora utilizando cuatro ventanas de comandos de Windows o cuatro terminales de Linux o MacOS, cada ventana ejecutará una instancia del programa.

## Código programado (sin Sockets seguros)

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

```

public class Token {
    static DataInputStream entrada;
    static DataOutputStream salida;
    static boolean inicio = true;
    static String ip;
    static int nodo;
    static long token;

    static class Worker extends Thread {
        @Override
        public void run() {
            try {
                ServerSocket servidor = new ServerSocket(20000 + nodo);
                Socket conexion = servidor.accept();
                entrada = new DataInputStream(conexion.getInputStream());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) throws InterruptedException,
    IOException {
        if (args.length != 2) {
            System.err.println("Se debe pasar como " +
                "parametros el numero de nodo y la " +
                "ip del siguiente nodo");
            System.exit(1);
        }

        nodo = Integer.parseInt(args[0]);
        ip = args[1];

        Worker w = new Worker();
        w.start();
        Socket conexion;

        while (true) {
            try {
                conexion = new Socket(ip, 20000 + (nodo+1)%4);
                break;
            } catch (IOException e) {
                Thread.sleep(500);
            }
        }

        salida = new DataOutputStream(conexion.getOutputStream());
        w.join();

        while (true) {
            if (nodo == 0) {
                if (inicio) {
                    inicio = false;
                    token = 1;
                } else {
                    token = entrada.readLong() + 1;
                    System.out.println(nodo + " " + token);
                }
            }
        }
    }
}

```

```

    }
} else {
    token = entrada.readLong() + 1;
    System.out.println(nodo + " " + token);
}

if (nodo == 0 && token >= 1000)
    break;

salida.writeLong(token);

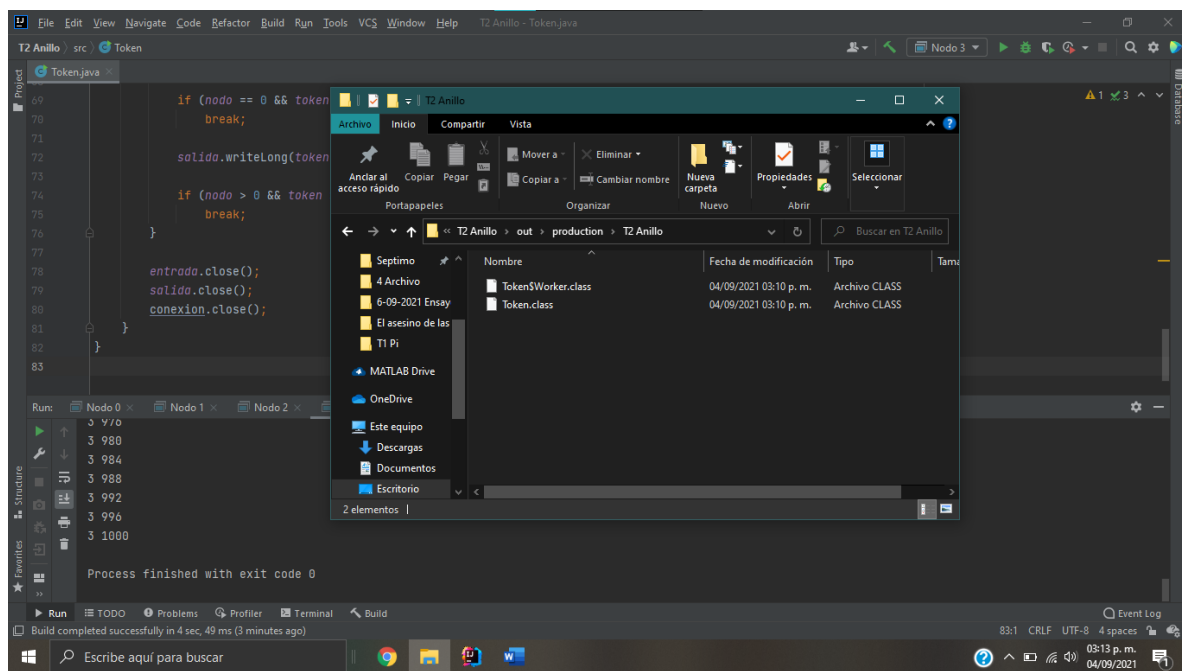
if (nodo > 0 && token >= 998)
    break;
}

entrada.close();
salida.close();
conexion.close();
}
}

```

## Resultados

Primero mostramos los class generados en el explorador de archivos:



## Código con Sockets seguros

Las diferencias más notables son las líneas donde se instanciaba el Socket de conexión y el de servidor, así como poner las cuatro propiedades necesarias para que lea los certificados y los acepte de forma correcta:

```

import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocketFactory;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Token {
    static DataInputStream entrada;
    static DataOutputStream salida;
    static boolean inicio = true;
    static String ip;
    static int nodo;
    static long token;

    static class Worker extends Thread {
        @Override
        public void run() {
            try {
                SSLServerSocketFactory socket_factory =
(SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
                ServerSocket servidor =
socket_factory.createServerSocket(20000 + nodo);
                Socket conexion = servidor.accept();
                entrada = new DataInputStream(conexion.getInputStream());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) throws InterruptedException,
IOException {
        System.setProperty("javax.net.ssl.keyStore",
"keystore_servidor.jks");
        System.setProperty("javax.net.ssl.keyStorePassword", "1234567");
        System.setProperty("javax.net.ssl.trustStore",
"keystore_cliente.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "123456");

        if (args.length != 2) {
            System.err.println("Se debe pasar como " +
"parametros el numero de nodo y la " +
"ip del siguiente nodo");
            System.exit(1);
        }

        nodo = Integer.parseInt(args[0]);
        ip = args[1];

        Worker w = new Worker();
        w.start();
        SSLSocketFactory cliente = (SSLSocketFactory)
SSLSocketFactory.getDefault();
        Socket conexion;

```

```

while (true) {
    try {
        conexion = cliente.createSocket(ip, 20000 + (nodo+1)%4);
        break;
    } catch (IOException e) {
        Thread.sleep(500);
    }
}

salida = new DataOutputStream(conexion.getOutputStream());
w.join();

while (true) {
    if (nodo == 0) {
        if (inicio) {
            inicio = false;
            token = 1;
        } else {
            token = entrada.readLong() + 1;
            System.out.println(nodo + " " + token);
        }
    } else {
        token = entrada.readLong() + 1;
        System.out.println(nodo + " " + token);
    }

    if (nodo == 0 && token >= 1000)
        break;

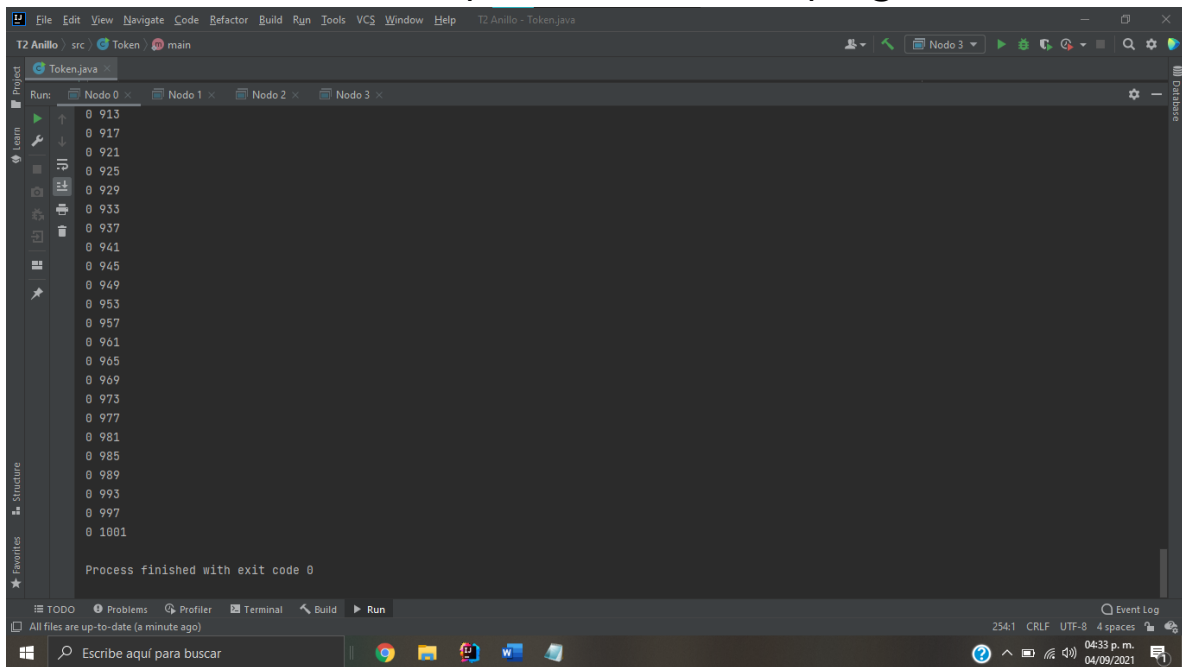
    salida.writeLong(token);

    if (nodo > 0 && token >= 998)
        break;
}

entrada.close();
salida.close();
conexion.close();
}
}

```

# Resultados Capturas al correr el programa



T2 Anillo - Token.java

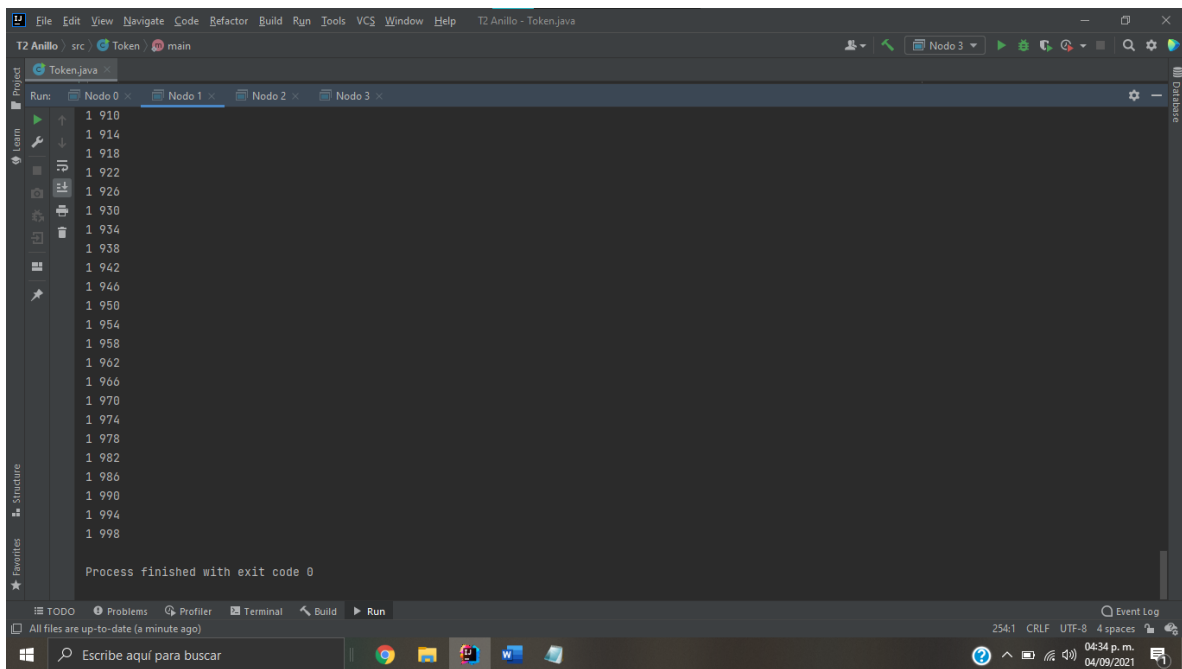
Run: Token.java x

0 913  
0 917  
0 921  
0 925  
0 929  
0 933  
0 937  
0 941  
0 945  
0 949  
0 953  
0 957  
0 961  
0 965  
0 969  
0 973  
0 977  
0 981  
0 985  
0 989  
0 993  
0 997  
0 1001

Process finished with exit code 0

254:1 CRLF UTF-8 4 spaces

04:33 p. m.  
04/09/2021



T2 Anillo - Token.java

Run: Token.java x

1 910  
1 914  
1 918  
1 922  
1 926  
1 930  
1 934  
1 938  
1 942  
1 946  
1 950  
1 954  
1 958  
1 962  
1 966  
1 970  
1 974  
1 978  
1 982  
1 986  
1 990  
1 994  
1 998

Process finished with exit code 0

254:1 CRLF UTF-8 4 spaces

04:34 p. m.  
04/09/2021

```
Run: Nodo 0 x Nodo 1 x Nodo 2 x Nodo 3 x
2 911
2 915
2 919
2 923
2 927
2 931
2 935
2 939
2 943
2 947
2 951
2 955
2 959
2 963
2 967
2 971
2 975
2 979
2 983
2 987
2 991
2 995
2 999
Process finished with exit code 0
```

```
Run: Nodo 0 x Nodo 1 x Nodo 2 x Nodo 3 x
3 912
3 916
3 920
3 924
3 928
3 932
3 936
3 940
3 944
3 948
3 952
3 956
3 960
3 964
3 968
3 972
3 976
3 980
3 984
3 988
3 992
3 996
3 1000
Process finished with exit code 0
```

## Conclusiones

Aunque el código que se mostró aquí es un poco diferente ya que los nodos del 1 al 3 muestran una excepción porque el primer nodo se desconecta, además, faltaron las líneas para cerrar los Streams así como el Socket, sin embargo es interesante ver que se puede asimilar a una topología de anillo con el código.