**Proiect RC - Comunicare în Inel**

**Șomodi Denise, Robu Raluca   
  
 Grupa : 30233**

**Aprilie 2025**

# Cuprins

1. Introducere  
2. Prezentarea topologiei  
3. Justificarea limbajului de programare ales  
4. Implementarea aplicației  
5. Testare și capturi de ecran  
6. Bibliografie  
7. Anexă

# Introducere

În cadrul acestui proiect am implementat un scenariu de comunicare între mai multe noduri, utilizând o topologie de tip **inel (ring topology)**, în care fiecare nod este conectat direct la următorul, formând astfel un ciclu închis. Comunicarea se realizează prin utilizarea socket-urilor TCP, fiecare nod având rolul de a primi, procesa și transmite mai departe informația către următorul participant din inel.

Scopul principal al proiectului este de a simula un proces de comunicare distribuit, în care fiecare nod colaborează pentru propagarea unui mesaj numeric. Fiecare nod din sistem primește o valoare întreagă, o incrementează cu o unitate și o transmite mai departe. Procesul continuă în această manieră ciclică până când valoarea atinge pragul de **100**, moment în care comunicarea se oprește, iar nodurile încetează să mai trimită date.

Prin realizarea acestui proiect, se urmărește aprofundarea cunoștințelor legate de:

* Programarea rețelelor cu socket-uri TCP.
* Organizarea comunicației între procese independente.
* Sincronizarea și gestionarea transmiterii controlate a mesajelor într-un sistem distribuit.

# Prezentarea topologiei

Topologia de rețea de tip **inel (ring topology)** este un model de comunicație în care fiecare nod este conectat la exact alte două noduri, formând împreună o structură circulară. În această arhitectură, datele sunt transmise de la un nod la următorul, într-o singură direcție (unidirecțional) sau în ambele direcții (bidirecțional, în cazul inelului dublu). În proiectul de față, comunicarea este implementată **unidirecțional**, astfel încât fiecare nod trimite mesajul către următorul nod, iar ultimul nod transmite mai departe către primul nod, completând ciclul.

Sistemul este alcătuit din **trei noduri**, fiecare având același comportament logic:

* Primește o valoare întreagă de la nodul anterior.
* Incrementează valoarea primită cu 1.
* Trimite noua valoare către următorul nod din inel.

Procesul continuă până când valoarea numerică atinge pragul de **100**, moment în care comunicarea încetează. Oprirea este controlată, astfel încât toate nodurile se opresc corect și sincronizat după ce mesajul cu valoarea 100 parcurge o tură completă a inelului.

### ****Avantajele utilizării topologiei de tip inel****:

* **Performanță constantă:** Comunicarea între noduri are loc într-o ordine bine definită, ceea ce reduce riscul coliziunilor de date.
* **Ușurință în detectarea erorilor:** Într-un inel, dacă un nod sau o conexiune eșuează, această problemă poate fi localizată ușor, deoarece există o secvență clară de transmitere a datelor.
* **Simplu de configurat și de extins:** Noduri suplimentare pot fi adăugate în rețea fără a afecta semnificativ structura sau logica aplicației, necesitând doar conectarea la două noduri existente.
* **Flux ordonat al mesajelor:** Fiecare nod primește acces la canalul de comunicare pe rând, ceea ce previne suprapunerea sau coliziunea pachetelor.

### ****Dezavantaje ale topologiei de tip inel****:

* **Dependență ridicată între noduri:** Dacă unul dintre noduri sau conexiunea acestuia eșuează (în cazul implementării unidirecționale), întregul flux de date este întrerupt.
* **Timpul de transmitere poate crește odată cu numărul nodurilor:** Deoarece mesajele trebuie să treacă prin fiecare nod, durata unei transmiteri crește proporțional cu dimensiunea inelului.
* **Dificultăți la debug în cazul unui inel de dimensiuni mari:** Cu cât sunt mai multe noduri, cu atât este mai complex să identifici rapid sursa unei erori, fără mecanisme suplimentare de monitorizare.

A pink ovals on a black background

AI-generated content may be incorrect.

*Fig. 1. Reprezentarea topologiei de tip inel utilizată în proiect.*

# 3. Justificarea limbajului de programare ales

Pentru realizarea acestui proiect am ales limbajul de programare **Java**, datorită multiplelor avantaje pe care acesta le oferă în dezvoltarea aplicațiilor de rețea și a proiectelor orientate pe comunicație între procese. Una dintre cele mai importante caracteristici ale Java este portabilitatea, având la bază principiul „Write Once, Run Anywhere” (WORA). Această proprietate permite rularea aplicației pe orice platformă care dispune de o mașină virtuală Java (JVM), fără a necesita modificări ale codului sursă, ceea ce contribuie la o dezvoltare flexibilă și eficientă.

Un alt motiv esențial care a stat la baza alegerii acestui limbaj este suportul puternic pentru programarea rețelei, oferit prin pachetul java.net, care include clase precum Socket și ServerSocket. Acestea facilitează implementarea comunicației TCP între noduri și simplifică procesul de creare a aplicațiilor de tip client-server, permițând o gestionare clară a conexiunilor și a fluxului de date. Astfel, Java oferă un cadru robust pentru dezvoltarea de aplicații distribuite, în care fiabilitatea transmiterii mesajelor și sincronizarea între participanți sunt aspecte esențiale.

De asemenea, Java este un limbaj orientat pe obiect, ceea ce permite o organizare modulară și clară a codului, favorizând reutilizarea componentelor și întreținerea facilă a aplicației. Gestionarea automată a memoriei prin mecanismul de garbage collection reduce riscul apariției scurgerilor de memorie și contribuie la stabilitatea generală a aplicației, mai ales în scenarii unde se lucrează cu multiple conexiuni și procese active.

Având în vedere aceste aspecte, Java s-a dovedit a fi o alegere potrivită pentru implementarea acestui proiect, oferind atât fiabilitate și siguranță în transmiterea datelor, cât și un mediu de dezvoltare prietenos și eficient pentru realizarea unei aplicații de comunicație în topologie de tip inel.

# 4. Implementarea aplicației

Pentru realizarea aplicației am utilizat o singură clasă, numită **RingNode**, care implementează logica de comunicare între nodurile participante în inel. Această clasă este instanțiată de trei ori, fiecare instanță având parametri diferiți pentru portul de ascultare, adresa următorului nod, portul următorului nod și un flag boolean care indică dacă nodul respectiv este inițiatorul comunicației.

În cadrul constructorului clasei RingNode, fiecare nod își setează propriile valori pentru aceste atribute și decide dacă trebuie să inițieze transmiterea mesajului, în funcție de parametrul isInitiator. Dacă nodul este inițiator, acesta trimite prima valoare (1) către următorul nod. În caz contrar, nodul rămâne în stare de ascultare, așteptând să primească o valoare de la nodul precedent.

A black screen with white text

AI-generated content may be incorrect.

*Fig. 2. Constructorul clasei RingNode.*

Pentru a implementa ascultarea mesajelor, fiecare nod pornește un server TCP pe portul său, folosind un ServerSocket. Nodul acceptă conexiuni și citește valorile transmise printr-un BufferedReader. Valoarea primită este verificată, incrementată și apoi transmisă către următorul nod, cu excepția cazului în care valoarea a atins limita de 100.

A screen shot of a computer program

AI-generated content may be incorrect.

*Fig. 3. Metoda start() din clasa RingNode*

Logica principală pentru oprirea comunicației este implementată în metoda listen(). Atunci când un nod primește valoarea 100, afișează mesajul „Reached 100, stopping...” și trimite această valoare o ultimă dată către următorul nod, pentru a-l informa că procesul s-a încheiat. Apoi ciclul se oprește prin ieșirea din bucla while.

Transmiterea mesajelor către următorul nod se face cu ajutorul metodei sendValue(). Aceasta deschide un socket către adresa și portul următorului nod și trimite valoarea utilizând un PrintWriter.

A screen shot of a computer code

AI-generated content may be incorrect.

*Fig. 4. Metoda sendValue(), care trimite valoarea către următorul nod din inel*

Pentru rularea corectă a aplicației, fiecare nod trebuie pornit cu parametrii corespunzători:  
Node2: 5001 127.0.0.1 5002 false  
Node3: 5002 127.0.0.1 5000 false  
Node1: 5000 127.0.0.1 5001 true

# 5. Testare și capturi de ecran

Pentru a valida funcționarea corectă a aplicației, am realizat testarea sistemului atât prin observarea outputului din terminal pentru fiecare nod, cât și prin monitorizarea traficului de rețea utilizând instrumentul Wireshark.

**5.1. Output pentru fiecare nod**

În această imagine este prezentat output-ul nodului inițiator, care are rolul de a începe procesul de comunicare prin transmiterea primei valori (1) către următorul nod din inel. Se observă cum valoarea este incrementată la fiecare pas și continuă să circule între noduri până când ajunge la pragul de 100.

**A screenshot of a computer

AI-generated content may be incorrect.**

*Fig. 5. Output Node1*

Imaginea prezintă secvența de mesaje procesate de Node2, care are rolul de a primi valoarea de la nodul anterior și de a o transmite către următorul nod din topologia de tip inel. Se observă alternanța mesajelor primite și trimise.

A screenshot of a computer

AI-generated content may be incorrect.

*Fig. 6. Output Node2*

**A screenshot of a computer

AI-generated content may be incorrect.**

*Fig. 7. Output Node3*

**5.2. Wireshark**

În următoarea imagine este prezentată capturarea pachetelor de rețea între cele trei noduri, folosind Wireshark. Filtrul aplicat (tcp.port == 5000 || tcp.port == 5001 || tcp.port == 5002) permite vizualizarea clară a comunicației TCP între porturile pe care rulează fiecare nod. Se poate observa direcția traficului (Source și Destination), precum și detalii din antetul TCP, cum ar fi secvențele SYN, ACK sau PSH, care indică atât procesul de stabilire a conexiunilor (TCP handshake), cât și transmiterea efectivă a mesajelor în timpul funcționării aplicației. Acest lucru confirmă că datele circulă corect între noduri în cadrul topologiei de tip inel implementate.

A screenshot of a computer

AI-generated content may be incorrect.

*Fig. 8. Captură Wireshark cu traficul filtrat pe porturile utilizate de noduri.*

A screenshot of a computer

AI-generated content may be incorrect.

*Fig. 9. Detalii ale unui pachet TCP*

Captura de mai sus prezintă analiza detaliată a unui pachet TCP în timpul funcționării aplicației. Se pot observa informațiile din antetul protocolului TCP, cum ar fi:

* **Source Port:** 5002
* **Destination Port:** 50212
* **Sequence Number:** 1
* **Acknowledgment Number:** 5
* **Length:** 0 (nu are date utile în payload, fiind un pachet de control - ACK)

În partea de jos este afișată reprezentarea hexazecimală și ASCII a pachetului, unde fiecare octet poate fi analizat individual. Această vizualizare permite confirmarea structurii pachetului și a valorilor din antet, demonstrând funcționarea corectă a comunicației TCP în cadrul aplicației.

# 6. Bibliografie

* https://www.lenovo.com/ca/en/glossary/what-is-ring-topology/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOoqyNHLE616-7Gxs3VUTV-y7ryRyykIzBFBamg6P6Q9kAHyU7akv
* <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-ring-topology/>
* <https://www.geeksforgeeks.org/socket-programming-in-java/>

# 7. Anexă

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
public class RingNode {  
 private int listenPort;  
 private String nextHost;  
 private int nextPort;  
 private boolean isInitiator;  
  
 public RingNode(int listenPort, String nextHost, int nextPort, boolean isInitiator) {  
 this.listenPort = listenPort;  
 this.nextHost = nextHost;  
 this.nextPort = nextPort;  
 this.isInitiator = isInitiator;  
 }  
  
 public void start() {  
 new Thread(this::listen).start();  
  
 if (isInitiator) {  
 try {  
 Thread.*sleep*(1000);  
 sendValue(1);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void listen() {  
 try (ServerSocket serverSocket = new ServerSocket(listenPort)) {  
 System.*out*.println("Listening on port " + listenPort);  
 while (true) {  
 try (Socket socket = serverSocket.accept();  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {  
  
 String line = in.readLine();  
 int receivedValue = Integer.*parseInt*(line);  
 System.*out*.println("Received: " + receivedValue + " on port " + listenPort);  
  
 if (receivedValue >= 100) {  
 System.*out*.println("Reached 100, stopping...");  
 sendValue(receivedValue);  
 break;  
 }  
  
 sendValue(receivedValue + 1);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void sendValue(int value) {  
 try (Socket socket = new Socket(nextHost, nextPort);  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {  
 out.println(value);  
 System.*out*.println("Sent: " + value + " to " + nextHost + ":" + nextPort);  
 } catch (IOException e) {  
 System.*out*.println("Ending communication here.");  
 }  
 }  
  
 public static void main(String[] args) {  
 if (args.length != 4) {  
 return;  
 }  
  
 int listenPort = Integer.*parseInt*(args[0]);  
 String nextHost = args[1];  
 int nextPort = Integer.*parseInt*(args[2]);  
 boolean isInitiator = Boolean.*parseBoolean*(args[3]);  
  
 RingNode node = new RingNode(listenPort, nextHost, nextPort, isInitiator);  
 node.start();  
 }  
}