```java
import java.net.*;
import java.net.UnknownHostException;
import java.io.*;
import java.nio.ByteBuffer;
import java.rmi.*;
import java.rmi.ConnectException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;


public class StorageNode implements Runnable {
		ScannerPorte scanner;
		StorageNode_TCP TCP;
		String bootstrapServerIp;
		static int node_number = 0;
		public StorageNode (ScannerPorte scanner) {
			this.scanner = scanner;
			this.bootstrapServerIp = null;
		}

		public StorageNode(ScannerPorte scanner, String bootstrapServerIp) {
			this.scanner = scanner;
			this.bootstrapServerIp = bootstrapServerIp;
		}

		public void run () {
			int file_id;
			String predIP = "", succIP = "", nodeIp = null; //questi due indirizzi ip appartengono
rispettivamente al mio predecessore e il mio successore
			int predPort = 0, succPort = 0; //analoghi agli indirizzi a predIP e succIP, ma
identificano le porte tcp da contattare
			Hashtable<Integer, String> nodeTable = new Hashtable<Integer, String>();
			Hashtable<Integer, String> dataTable = new Hashtable <Integer, String>();
			//genero il mio indirizzo ip

			if (bootstrapServerIp == null) //versione locale
				{nodeIp = String.valueOf((int) (Math.random() * 256)) + "."
					+ String.valueOf((int) (Math.random() * 256)) + "."
					+ String.valueOf((int) (Math.random() * 256)) + "."
					+ String.valueOf((int) (Math.random() * 256));
				} else //versione di rete
				try {
					nodeIp = InetAddress.getLocalHost().getHostAddress();

				} catch (UnknownHostException e) {
					e.printStackTrace();
				}

			//creo già le Socket d'ingresso su cui il nodo verrà contattato
			//Socket UDP di servizio: serve per esser contattato quando si è un nodo di bootstrap
			DatagramSocket UDPServiceSocket = scanner.UDPSocket();

			if (UDPServiceSocket == null) {
				System.out.println("Tentativo di creazione socket UDP fallito...");
				return ;
			}

			//Socket TCP di servizio: server per esser contattato da altri nodi
			ServerSocket TCPServiceSocket = scanner.TCPSocket();
```

```
                        if (TCPServiceSocket == null) {
                                System.out.println("Tentativo di creazione socket TCP fallito...");
                                return ;
                        }

                        int UDPport = UDPServiceSocket.getLocalPort();
                        int TCPport = TCPServiceSocket.getLocalPort();
                        nodeIp = nodeIp + " UDPport=" + UDPport + " TCPport=" + TCPport;

                        try { //mi connetto al server tramite RMI
                                Remote RemoteObject;
                                BootstrapServer_Interface serverObject;
                                Registry r;

                                if (bootstrapServerIp != null) //versione di rete
                                        {r = LocateRegistry.getRegistry(bootstrapServerIp, 10001);
                                        } else //versione locale
                                        r = LocateRegistry.getRegistry(10001);

                                serverObject = (BootstrapServer_Interface) r.lookup("BOOTSTRAP-SERVER");

                                //Ottengo l'ip del Bootstrap Node a cui connettermi
                                System.out.println("Il nodo " + Thread.currentThread() + " si mette in
attesa");

                                //Se continuo, allora ho ottenuto la lock dal BootstrapServerMain
                                String ipBN = serverObject.start_join_node(nodeIp); //qui non c'è
distinzione tra locale e rete

                                if (ipBN.equals("-1")) {
                                        System.out.println("Creazione del nodo " +
Thread.currentThread() + " fallito");
                                        return ;
                                }

                                System.out.println("Inizio inserimento del nodo " + Thread.currentThread()
+ " nell'anello");
                                //se il nuovo nodo è un novo di bootstrap

                                if (ipBN.contains(" bootstrapnode")) {
                                        StorageNode_KeepAlive KA;

                                        if (bootstrapServerIp != null)        //versione di rete
                                                KA = new StorageNode_KeepAlive(nodeIp,
bootstrapServerIp, scanner); //si occuperà di inviare al BootstrapServer_KeepAlive i propri segnali
                                        else
                                                KA = new StorageNode_KeepAlive(nodeIp, scanner);

                                        KA.start();

                                        ipBN = ipBN.substring(0, ipBN.indexOf(" bootstrapnode"));
//elimino la stringa "bootstrapnode" dalla stringa ipBN
                                }

                                //se non sono il primo nodo mi ricavo il mio node_number
                                if (!ipBN.equals("1")) {
                                        int end_node_number = ipBN.indexOf(" ip=");
                                        node_number = Integer.parseInt(ipBN.substring(0,
end_node_number));
                                        ipBN = ipBN.substring(end_node_number + 4, ipBN.length());
//elimino il nodenumber dalla stringa ipBN
```

```
                } else
                node_number = 1;

        //Se sono il primo nodo leggo tutti i dati che saranno presenti nel
sistema
        if (node_number == 1) {
                NodeFileReader reader = new NodeFileReader
("textfile.txt");
                dataTable = reader.read();

                if (dataTable.isEmpty())
                        System.out.println("Errore: tabella dati vuota");
        }

        //Genero il mio identificatore tramite la SHA-1
        MessageDigest md = MessageDigest.getInstance("SHA1");

        md.update(nodeIp.getBytes());

        byte[] output = md.digest();

        ByteBuffer bb = ByteBuffer.wrap(output);

        file_id = Math.abs((int) bb.getLong()) % (2 ^ 125);

        //se non sono il primo nodo devo contattare il nodo di bootstrap
        if (!ipBN.equals("1")) { //ottengo porta UDP del  bootstrap
                int start = ipBN.indexOf("UDPport=");
                int end = ipBN.indexOf(" TCPport=");
                String sPort = ipBN.substring(start + 8, end);
                int bootStrapNodeport = Integer.parseInt(sPort);
                //ottengo l'indirizzo IP del nodo di bootstrap
                ipBN = ipBN.substring(0, start - 1);
                boolean alreadyexist;

                do {
                        alreadyexist = false;
                        ByteArrayOutputStream bout = new
ByteArrayOutputStream( );

                        DataOutputStream dos = new DataOutputStream
(bout);

                        //la stringa type indica che è un nodo a
contattatare il nodo di bootstrap
                        String type = "node";
                        dos.writeUTF(type);
                        //viene inviato anche l'identificatore del
proprio nodo
                        //così facendo il nodo di bootstrap può
calcolare il nostro succ. e pred.
                        dos.writeInt(file_id);
                        //vengono inviati anche i propri indirizzi
per permettere di ricevere una risposta
                        dos.writeUTF(nodeIp);
                        byte [ ] data = bout.toByteArray();
                        InetAddress ia = InetAddress.getLocalHost
();

                        if (bootstrapServerIp != null) //versione
di rete
                        {ia = InetAddress.getByName(ipBN);}

                        DatagramPacket dp = new DatagramPacket
(data, data.length, ia, bootStrapNodeport);
```

```
bootstrap...

l'indirizzo ip + porta del predecessore e del successore...

vero e proprio

ByteArrayInputStream(dp.getData());

());

ObjectInputStream(bin);

esiste già un nodo con il mio stesso id

" + Thread.currentThread() + " incrementa il proprio id (" + file_id + ") dato che esiste già nell'anello");

oin.readObject();

oin.readObject();

oin.readObject();

presenti nella rete

<Integer, String>) oin.readObject();

attualmente esistente)

si inserisce un nodo

" IP=" + nodeIp, 1);

" IP=" + nodeIp, 0);

gestisce il multicast)

nodeTable aggiornata
```

```
//invio il pacchetto al nodo di

UDPServiceSocket.send(dp);
//...il quale mi risponderà con

//data ora potrà contenere il pacchetto

data = new byte[5000];
dp = new DatagramPacket(data, data.length);
UDPServiceSocket.receive(dp);
ByteArrayInputStream bin = new

bin = new ByteArrayInputStream(dp.getData

ObjectInputStream oin = new

predIP = (String) oin.readObject();

if (predIP.equals("-1")) //significa che

        {alreadyexist = true;
                System.out.println("Il nodo
                file_id += 1;
        } else {
                succIP = (String)

                predPort = (Integer)

                succPort = (Integer)

                //...e la tabella dei nodi

                nodeTable = (Hashtable

        }

    } while (alreadyexist);
}

//aggiungo il mio identificatore alla tabella
nodeTable.put(file_id, "id=" + file_id + " IP=" + nodeIp);

//creo il thread che si occupa del multicast (passandogli la tabella

//l'oggetto che gestisce il multicast aggiorna la tabella ogni volta che

StorageNode_Multicast multicast;

if (bootstrapServerIp != null)        //versione di rete
        multicast = new StorageNode_Multicast(nodeTable, "id=" + file_id +

else                                                    //versione locale
        multicast = new StorageNode_Multicast(nodeTable, "id=" + file_id +

multicast.start();

//creo il mio collegamento udp in ingresso (passandogli l'oggetto che

//l'oggetto del multicast serve per poter ottenere quando necessario la

StorageNode_UDP UDP;
```

```
                                        if (bootstrapServerIp != null)        //versione di rete
                                                UDP = new StorageNode_UDP(UDPServiceSocket, multicast, scanner, 1);
                                        else
                                                UDP = new StorageNode_UDP(UDPServiceSocket, multicast, scanner, 0);

                                        UDP.start();

                                        //se non sono il primo nodo connettermi al mio successore e predecessore
con tcp
                                        if (bootstrapServerIp != null)
                                                TCP = new StorageNode_TCP(Thread.currentThread(), TCPServiceSocket,
predIP, succIP, nodeIp, predPort, succPort, file_id, dataTable, node_number, multicast, scanner, 1);
                                        else
                                                TCP = new StorageNode_TCP(Thread.currentThread(), TCPServiceSocket,
predIP, succIP, nodeIp, predPort, succPort, file_id, dataTable, node_number, multicast, scanner, 0);

                                        try {
                                                        TCP.start();
                                                        TCP.join(); //leggere la descrizione dell'attributo
"chiamante" nella classe StorageNode_TCP per approfondimenti su questo punto

                                                } catch (InterruptedException e) {}

                                        //Aggiunta del nodo completata
                                        System.out.println("Inserimento del nodo " + Thread.currentThread() + "
completato");


                                        serverObject.end_join_node(nodeIp);

                                } catch (ConnectException e) {
                                        System.out.println("Connessione al server fallita per il nodo " +
Thread.currentThread());
                                        {
                                                UDPServiceSocket.close();

                                                try {
                                                                TCPServiceSocket.close();

                                                        } catch (IOException e1) {
                                                                e1.printStackTrace();
                                                        }
                                        }

                                } catch (Exception e) {
                                        e.printStackTrace();
                                        UDPServiceSocket.close();

                                        try {
                                                        TCPServiceSocket.close();

                                                } catch (IOException e1) {
                                                        e1.printStackTrace();
                                                }
                                }
                        }
                }
```