Gruppo 17

Università degli Studi di
Bologna Scuola di Ingegneria

Esercitazione 6
Java RMI

Corso di Reti di Calcolatori T

Daniele Magagnoli, Giulia Martina Bal,
Antonio Cassanelli, Gianmiriano Porrazzo

Anno Accademico 2021/2022

SPECIFICHE:

L’esercitazione chiedeva la realizzazione di un’applicazione C/S, con l’utilizzo di RMI, per svolgere operazioni su file di testo remoti.

- il progetto prevede un’interfaccia remotizzabile nella quale vengono definiti due metodi, invocabili da remoto dal client: conta righe ed elimina righe.
- Oltre all’interfaccia RemOp, è necessario creare una classe ServerImpl e una classe Client.
- Il ServerImpl è una classe che estende UnicastRemoteObject e implementa l’interfaccia RemOp; si occupa della realizzazione dei due metodi.
- Il Client è un filtro che propone all’utente i due servizi ciclicamente. Viene invocato specificando l’host e la sua porta (di default 1099)

INTERFACCIA REMOTIZZABILE E SERVER

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RemOp extends Remote{

    int conta_righe(String nomeFile, int parole)
            throws RemoteException;

    String elimina_riga(int riga, String nomeFile)
            throws RemoteException;
}
```

A destra viene riportata la classe ServerImpl che implementa l'interfaccia RemOp. L'interfaccia RemOp estende la classe Remote e definisce i due metodi conta righe ed elimina righe che verranno implementati dal ServerImpl. Il main() si occupa principalmente di far registrare il server all'rmiregistry attraverso la funzione rebind().

```java
public class ServerImpl extends UnicastRemoteObject implements RemOp {

    private static final long serialVersionUID = 1L;

    // Costruttore
    public ServerImpl() throws RemoteException {
        super();
    }
// Avvio del Server RMI
public static void main(String[] args) {

    final int REGISTRYPORT = 1099;
    String registryHost = "localhost";
    String serviceName = "RemOp";       //lookup name...

    // Registrazione del servizio RMI
    String completeName = "//" + registryHost + ":" + REGISTRYPORT + "/"
            + serviceName;
    try{
        ServerImpl serverRMI = new ServerImpl();
        Naming.rebind(completeName, serverRMI);
        System.out.println("Server RMI: Servizio \"" + serviceName
                + "\" registrato");
    }
    catch(Exception e){
        System.err.println("Server RMI \"" + serviceName + "\": "
                + e.getMessage());
        e.printStackTrace();
        System.exit(1);
    }
}
```

```java
public int conta_righe(String nomeF, int numw)throws RemoteException{
    int res=0,i=0,wc=0;
    BufferedReader buff;
    char ch;
    long start=0,end=0;//usato per controlli

    try{
        buff= new BufferedReader (new FileReader(nomeF));
    }catch (FileNotFoundException e){
        throw new RemoteException(e.toString());
    }
    System.out.println("Conto il numero di righe con almeno "+numw+" parole in "+nomeF +" \n");

    start=System.currentTimeMillis();
    try{
        boolean valid=true;
        while((i=buff.read())>=0){
            ch=(char) i;

            if(ch!='\n' && valid)
                valid =false;

            if(ch==' ' || ch== ',' || ch=='.' || ch==':' || (!valid && ch=='\n')  ){
                wc++;
            }
            if(ch=='\n'){
                valid=true;
                System.out.println(" Parole: "+wc+"\n");//trovo il newline
                if(wc>numw) res++;//se il numero di parole contate supera il minimo aumento
                wc=0;               //setto a zero il counter delle parole per iniziare una nuov
            }
        }
    }catch(IOException e){
        throw  new RemoteException(e.toString());
    }
    end=System.currentTimeMillis();
    System.out.println("---Tempo impiegato server "+(end-start)+"---\n");
    try{
        buff.close();
    }catch (IOException e){
        throw new RemoteException(e.toString());
    }
    return res;
    }
```

FUNZIONE CONTA RIGHE

La funzione conta righe si aspetta come parametri d'ingresso il nome del file e il numero minimo di parole per ogni riga. Controlla se il file passato esiste, altrimenti lancia una RemoteException. Attraverso un BufferedReader leggiamo carattere per carattere: se il carattere corrisponde a '\n' allora viene fatto il controllo che il numero di parole della riga sia maggiore dell'intero passato come argomento e aggiorno il numero di righe. Eseguendo dei test si è notato che questa soluzione è più veloce rispetto alla lettura riga per riga.

ALTERNATIVA:

```
public int conta_righe(String nomeFile, int parole) throws RemoteException{
     int res=0;
     BufferedReader buff;
        try{
            buff= new BufferedReader (new FileReader(nomeFile));
        }catch (FileNotFoundException e){
            throw new RemoteException(e.toString());
        }
        System.out.println("Conto il numero di righe con almeno "+
                           parole+" parole in "+nomeFile +" \n");
        String line;
        try {
            while((line = buff.readLine())!=null){
                if(!line.isEmpty()) {
                    String[] word = line.split("\\s+|,\\s*|\\.\\s*");
                    System.out.println(word.length);
                    if(word.length>=parole)
                        res++;
                }
            }

            buff.close();
        }catch(IOException e) {
            throw new RemoteException(e.toString());
        }

        return res;
}
```

Una soluzione alternativa, ma meno veloce, è utilizzare un BufferedReader per leggere riga per riga e con l'utilizzo di una split() ricavare il numero di parole presenti per ogni riga.

```java
public synchronized String elimina_riga(String nomeF, int nl)throws RemoteException{
    int att=1,i; //il file inizia a contare dalla linea 1
    String newN = nomeF.substring(0,(nomeF.length()-4))+"_mod.txt";
    String res;
    char ch;
    long start=0,end=0;
    File temp = new File(newN);

    BufferedWriter out;
    BufferedReader in=null;

    try {
        in = new BufferedReader (new FileReader(nomeF));
        out= new BufferedWriter(new FileWriter(temp));
    }catch (IOException e){
        throw new RemoteException(e.toString());
    }

    System.out.println("Elimino la riga numero "+nl+" da "+ nomeF);

    start=System.currentTimeMillis();
    try{
        while((i=in.read())>=0){
            ch=(char) i;

            if(ch!='\n'){
                if(att!=nl)
                    out.write(ch);
            }
            else if(ch =='\n'){
                if(att!=nl)
                    out.write('\n');
                att++;
            }
        }

        in.close();
        out.close();
        if (att<nl){
            throw new RemoteException("Il file remoto ha "+att+" righe, deve averne almento "+nl);
        }else{
            res=newN+" "+(att-1);
        }
    }catch (IOException e){
        throw new RemoteException(e.toString());
    }
    end=System.currentTimeMillis();
    System.out.println("---Tempo impiegato server "+(end-start)+" ---\n");

    return res;
```

FUNZIONE ELIMINA RIGA

Riceve come parametri d'ingresso il nome di un file remoto e il numero di riga da eliminare. Attraverso un BufferedReader e un BufferedWriter riusciamo a leggere carattere per carattere e scriverli sul nuovo file creato, tranne quelli appartenenti alla riga da eliminare. Se il file non esiste, la scrittura non va a buon fine o il numero di riga da eliminare eccede quelle totali, viene lanciata una RemoteException.

```java
public synchronized String elimina_riga(String nomeFile, int riga)
        throws RemoteException{
    int att=1;
    String res;//il file inizia a contare dalla linea 1
    String newN = nomeFile.substring(0,(nomeFile.length()-4))+"_modified.txt";
    File temp = new File(newN);
    BufferedWriter out;
    BufferedReader in;
    try {
        out= new BufferedWriter (new FileWriter(temp));
        in = new BufferedReader (new FileReader(nomeFile));
    }catch (IOException e){
        throw new RemoteException(e.toString());
    }

    System.out.println("Elimino la riga numero "+riga+" da "+ nomeFile);

    String line;

    try {
        while((line=in.readLine())!=null){
            if( att != riga)  //se la linea non è quella da eliminare la scrivo
                out.write(line+"\n");
            att++;
        }
        in.close();
        out.close();
    }catch(IOException e) {
        throw new RemoteException(e.toString());
    }

    if (att< riga){
        throw new RemoteException();
    }else{
        res= newN +" " +(att-2);
        System.out.println(res);
    }

    return res;
```

ALTERNATIVA:

Come nel caso della conta righe possiamo avere una soluzione che legge riga per riga.
In questo caso a differenza della conta righe questo metodo risulta più veloce rispetto alla lettura carattere per carattere.

```java
try{
    String completeName = "//" + registryHost + ":" + REGISTRYPORT + "/"
            + serviceName;
    RemOp serverRMI = (RemOp) Naming.lookup(completeName);
    System.out.println("ClientRMI: Servizio \"" + serviceName + "\" connesso");

    System.out.println("\nRichieste di servizio fino a fine file");

    String service;
    System.out.print("Servizio (C=Conta righe di un file , E=elimina righe di un file): ");

    /*ciclo accettazione richieste utente*/
    while ((service = stdIn.readLine()) != null){

        if(service.equals("C")){
            boolean valid = false;
            int parole = 0;
            String fileName=null;
            System.out.print("Nome del file?\n");

            while(valid!=true) {
                fileName = stdIn.readLine();
                if(!fileName.endsWith(".txt")) {
                    System.out.println("Il file passato non è un file di testo\n");
                    System.out.print("Nome del file?\n");
                    continue;
                }
                else valid=true;
            }
            valid=false;
            System.out.print("Numero di parole minimo per riga? ");
            while(valid!=true) {

                System.out.print("Numero di parole minimo per riga?\n");
                try {
                    parole = Integer.parseInt(stdIn.readLine());
                    valid=true;
                }catch(NumberFormatException e) {
                    System.out.print("Errore formato!\n");
                    valid=false;
                }
            }
            try {
                System.out.println("Sono state trovate "+serverRMI.conta_righe(fileName, parole)
                    + " righe con più di "+parole+" parole nel file "+ fileName);

            }catch(RemoteException e) {
                System.out.println("Errore nell'operazione conta righe! "+ e);
                System.out.println("Servizio (C=Conta righe di un file , E=elimina righe di un file): ");
                continue;
            }
```

CLIENT

Il Client deve chiedere un servizio remoto al Server al quale si connette attraverso la funzione lookup(). Una volta connesso il Client chiede ciclicamente all'utente il tipo di servizio che vuole svolgere. Nel caso il servizio sia conta righe (C),il Client controlla che il file sia un file di testo e che l'intero che rappresenta il numero minimo di parole sia valido. Invoca alla fine il metodo remoto conta righe e cattura la RemoteException nel caso di errori richiedendo all'utente un nuovo servizio.

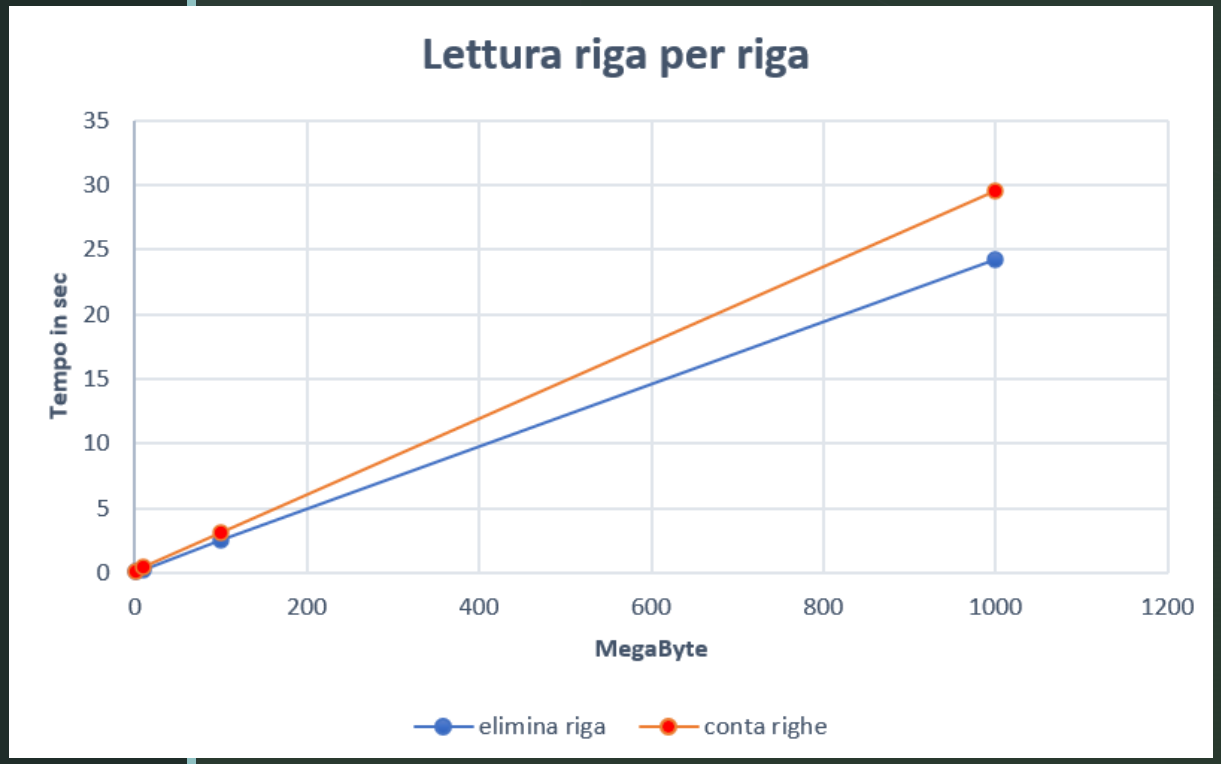
```
else if(service.equals("E")){
    int riga = 0;
    boolean valid = false;

    System.out.print("Nome del file?\n");
    String fileName = null;
    while(valid!=true) {
        fileName = stdIn.readLine();
        if(!fileName.endsWith(".txt")) {
            System.out.println("Il file passato non è un file di testo\n");
            System.out.print("Nome del file?\n");
            continue;
        }
        else valid=true;
    }
    valid=false;
    System.out.print("Riga da eliminare?\n");
    while (valid != true){
        try {
            riga = Integer.parseInt(stdIn.readLine());
            if(riga <=0){
                System.out.println("Riga non valida\n");
                System.out.print("Riga da eliminare?\n");
                continue;
            } else valid = true;

        }catch(NumberFormatException e) {
            System.out.println("Formato riga errato\n");
            System.out.print("Riga da eliminare?\n");
            continue;
        }
    }

    try {
        String risp[]= serverRMI.elimina_riga(fileName, riga).split(" ");
        System.out.println("Il numero di righe presenti nel nuovo file "+risp[0]+ " è "+ risp[1]+"\n");

    }catch(RemoteException e) {
        System.out.println("Errore nell'operazione elimina riga!");
        System.out.println("Servizio (C=Conta righe di un file , E=elimina righe di un file): ");
        continue;
    }

} // E=Elimina riga

else System.out.println("Servizio non disponibile");

System.out.print("Servizio (C=Conta righe di un file , E=elimina righe di un file): ");
/ while (!EOF), fine richieste utente
```

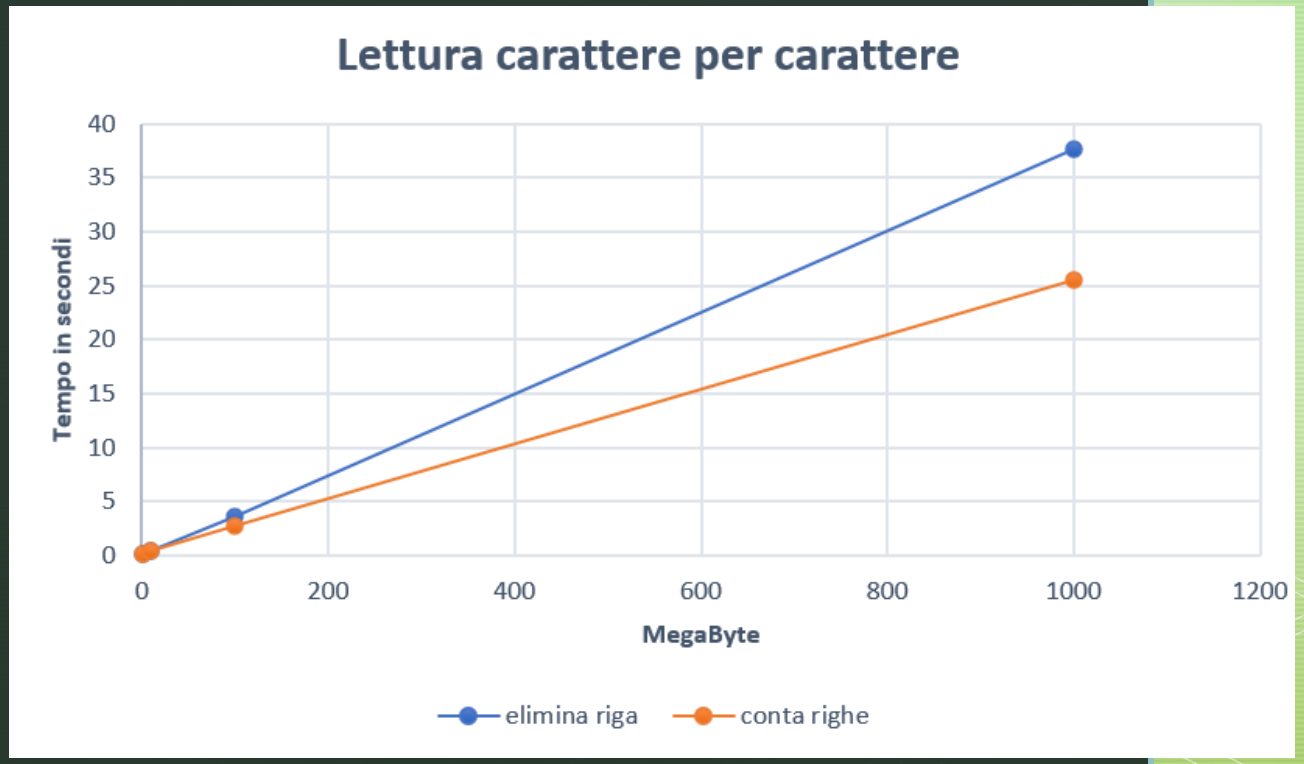
CLIENT

Nel caso il servizio sia elimina riga (E), il Client controlla sempre che il file sia un file di testo e che l'intero che rappresenta la riga sia valido. Invoca alla fine il metodo remoto elimina riga e cattura la RemoteException nel caso di errori richiedendo all'utente un nuovo servizio.

CONSIDERAZIONI FINALI:

Test eseguiti prendendo il tempo sul Client. I dati relativi alla conta righe sono stati presi considerando le righe con numero di parole minimo 1 (tutte le righe tranne quelle vuote) ; per l'elimina riga si è considerata l'eliminazione di una riga all'incirca nel mezzo del file.





GRAZIE
PER L’ATTENZIONE