Esercizio Anello e URL

Laboratorio di Programmazione di Rete A

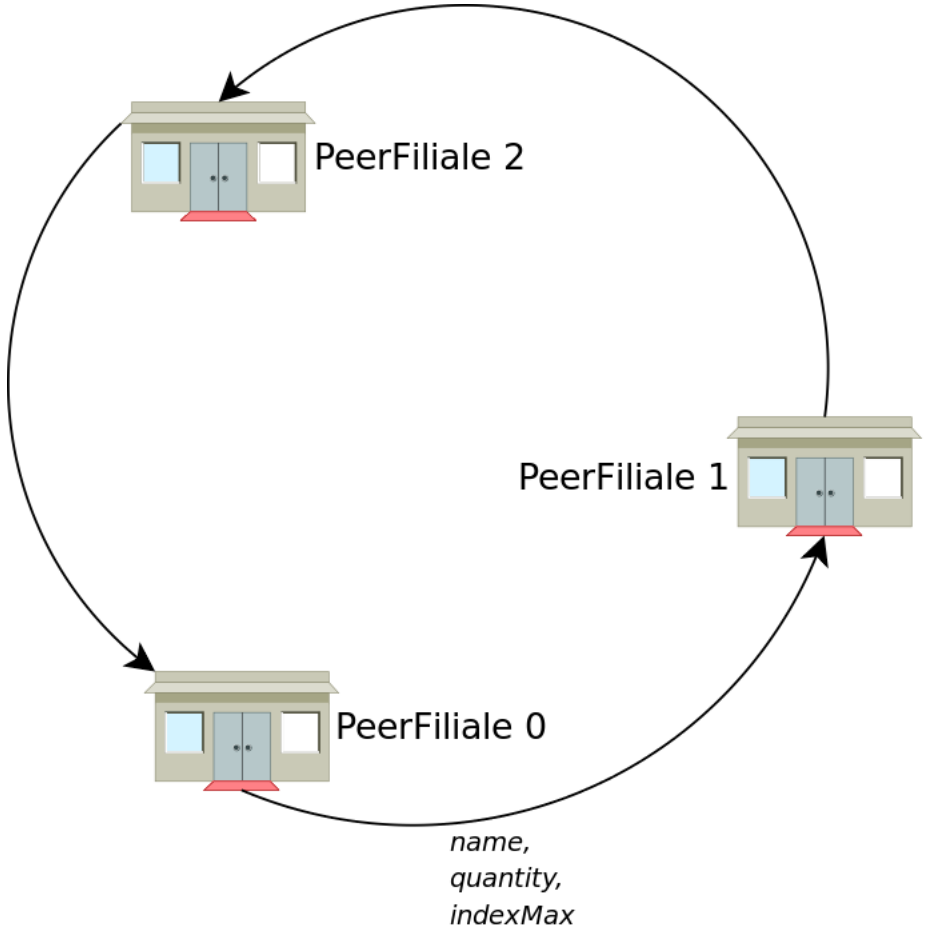
Esercitazione di Laboratorio

31/10/2007

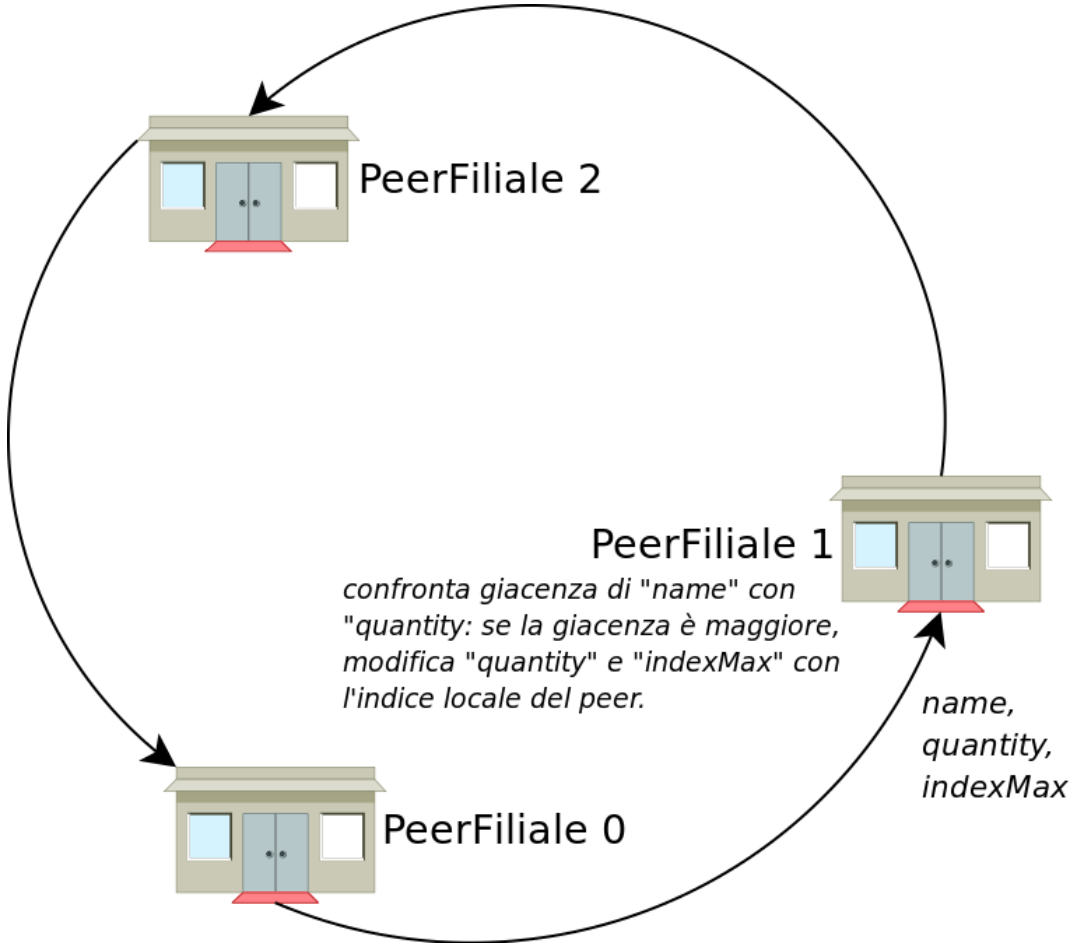
Esercizio su Anello UDP

- Si consideri una grande azienda produttrice di computer che possieda n filiali sparse sul territorio. Ogni filiale commercializza 10 modelli di computer prodotti dall'azienda.
- Si supponga che ogni filiale voglia conoscere, per ogni modello di computer, il nome della filiale che possiede nel suo magazzino il maggior numero di computer di quel modello.
- Si attivi, per ogni filiale, un programma PeerFiliale che memorizzi in un file le giacenze di computer presso quella filiale. Ogni PeerFiliale e' identificata da un indice univoco. I PeerFiliale sono interconnessi mediante una struttura logica ad anello (ogni PeerFiliale puo' solamente comunicare con il suo successore e predecessore nell'anello usando il protocollo UDP).
- La computazione del valore massimo per ogni modello e' innescata dal PeerFiliale di indice 0. Trascurare l'eventuale perdita di datagrammi.

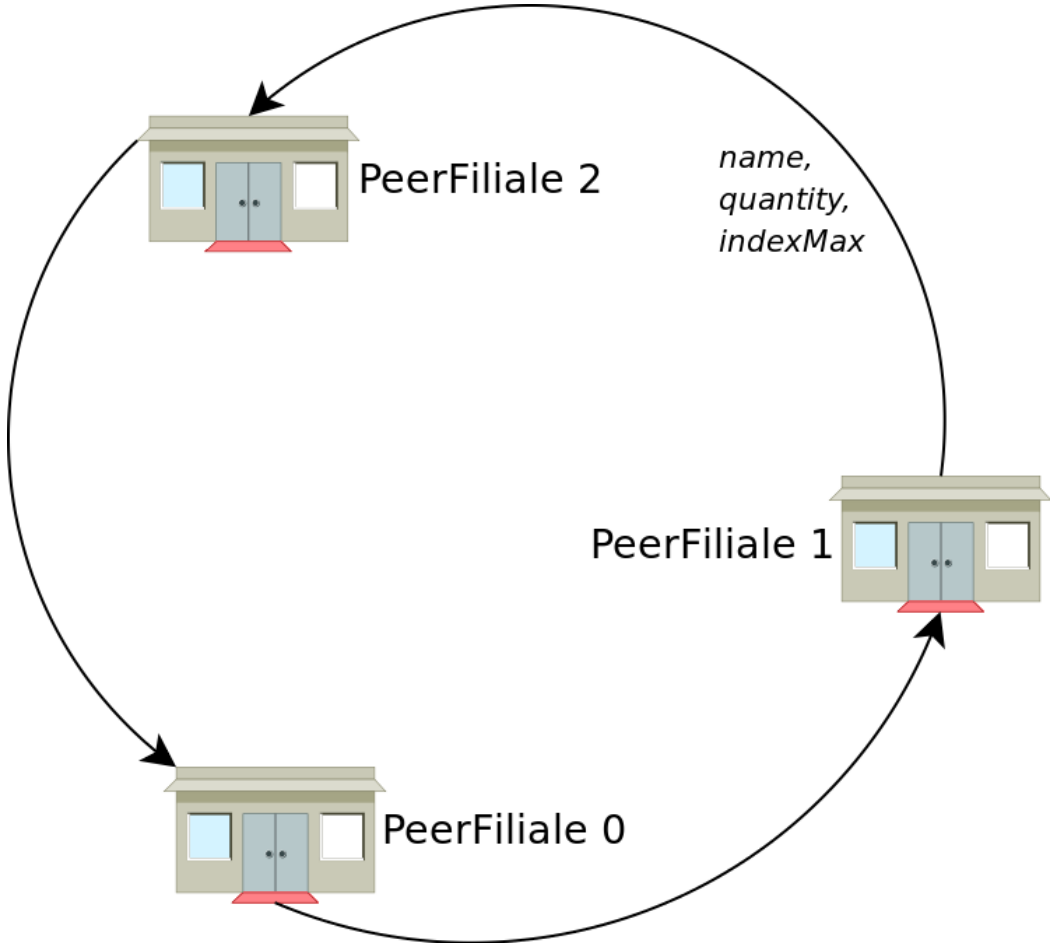
Primo Giro: Computazione del Massimo



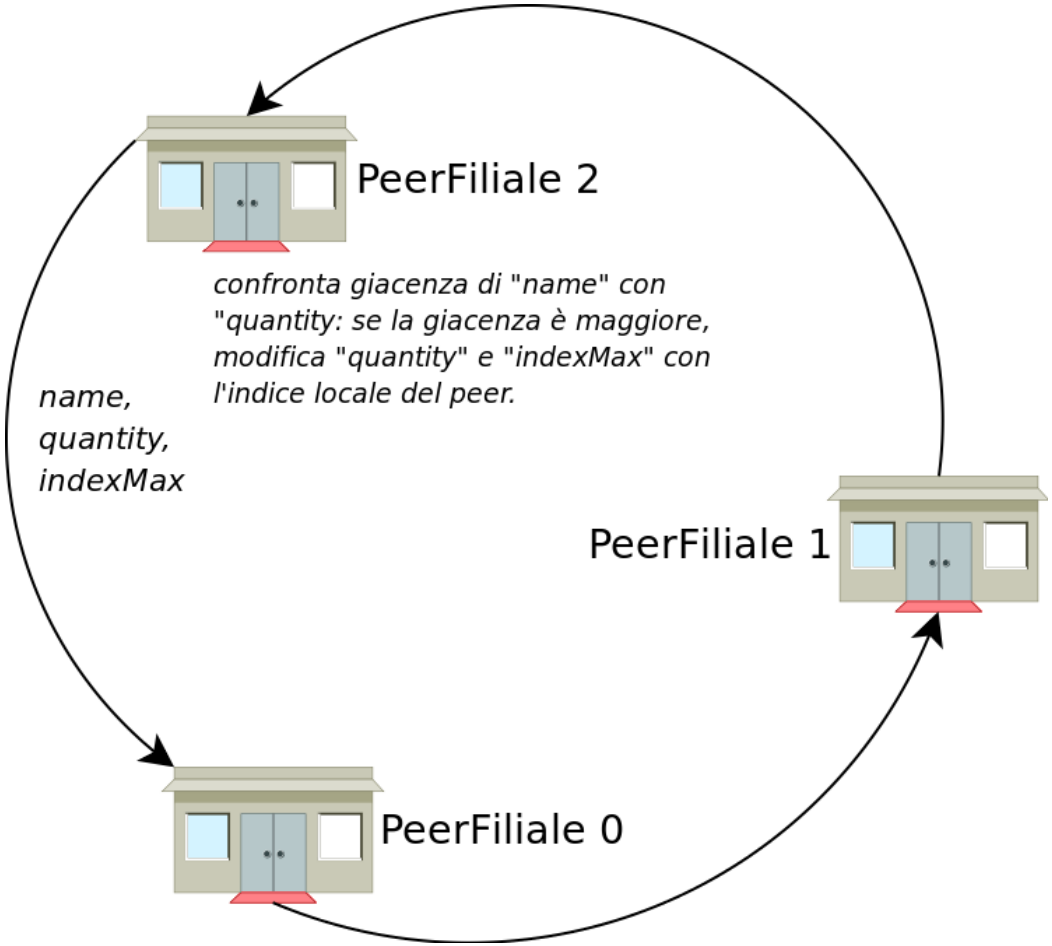
Primo Giro: Computazione del Massimo



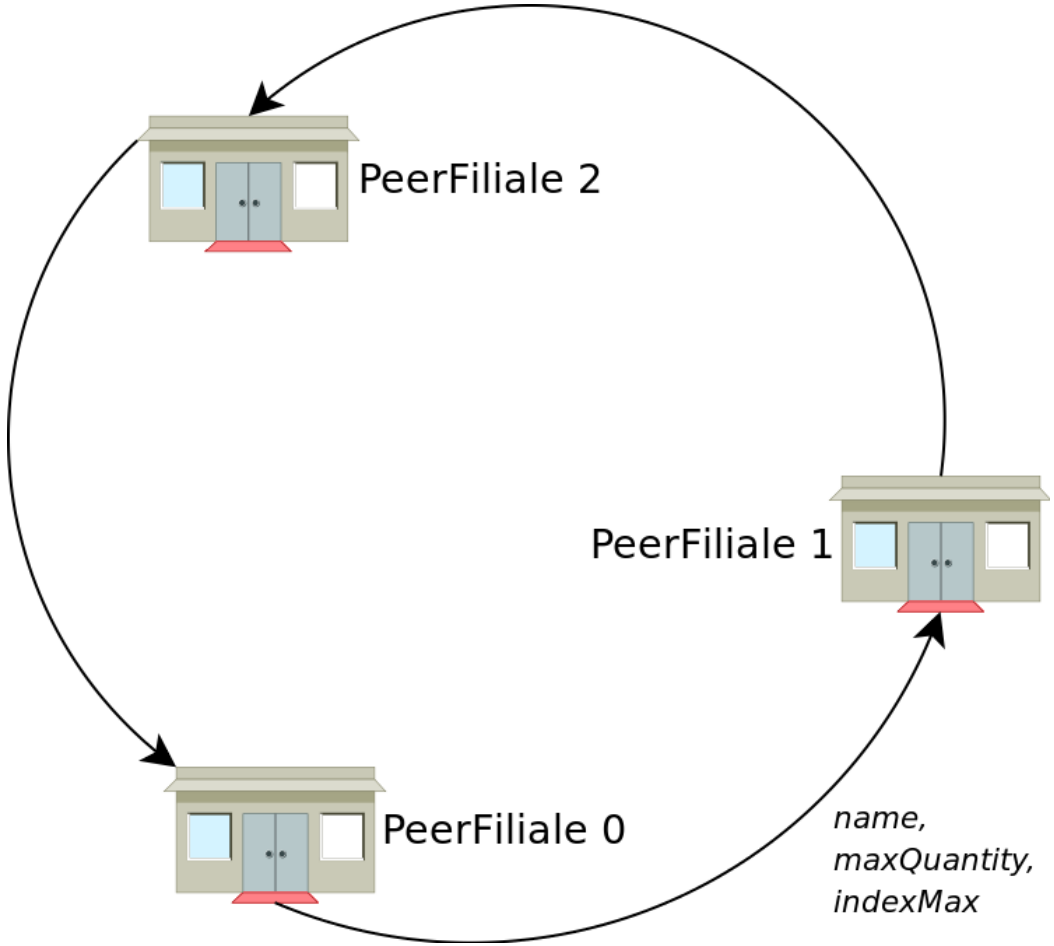
Primo Giro: Computazione del Massimo



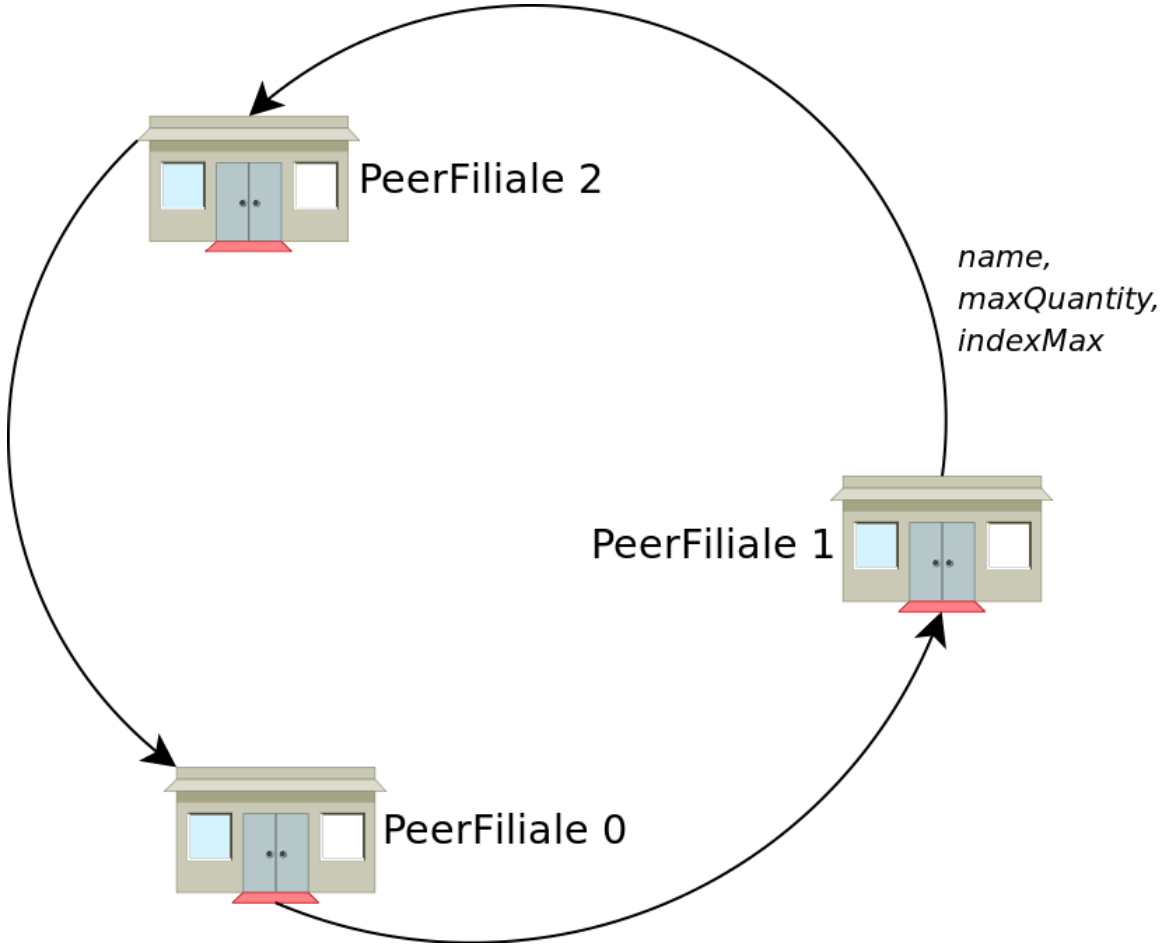
Primo Giro: Computazione del Massimo



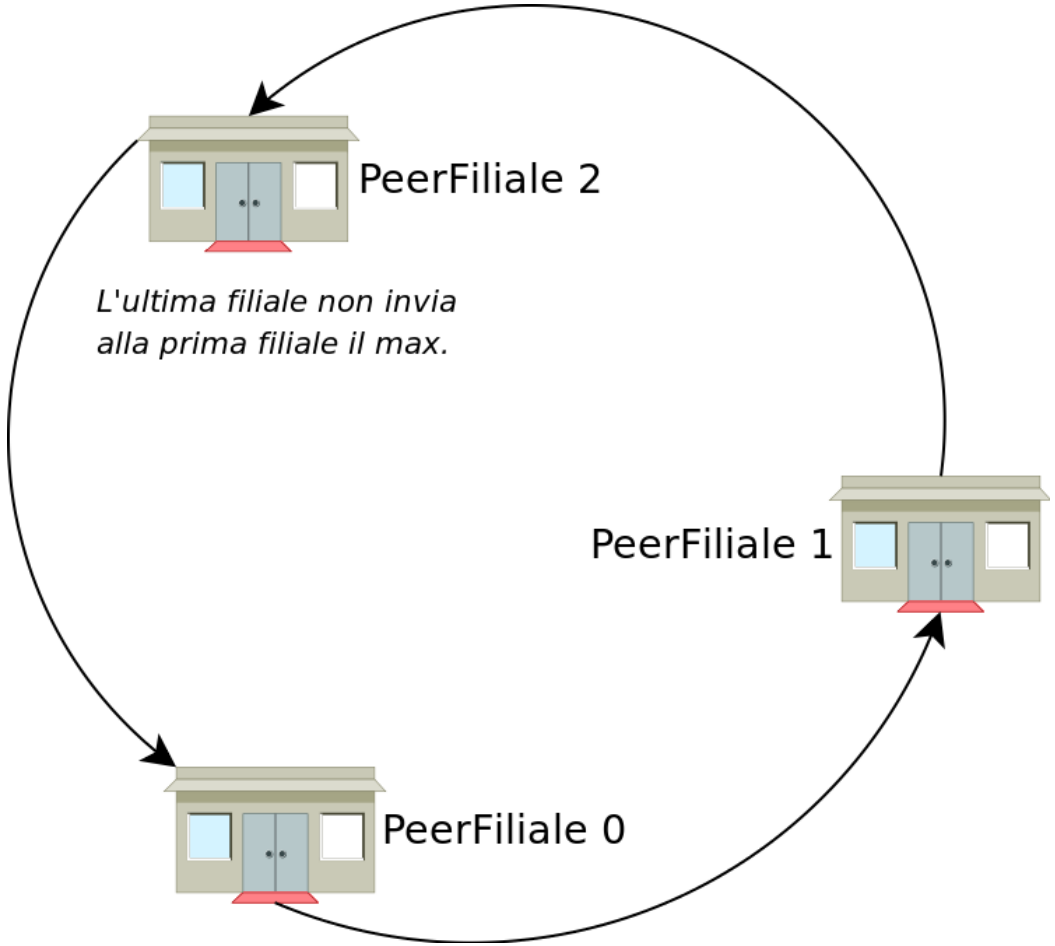
Secondo Giro: Comunicazione del Massimo



Secondo Giro: Comunicazione del Massimo

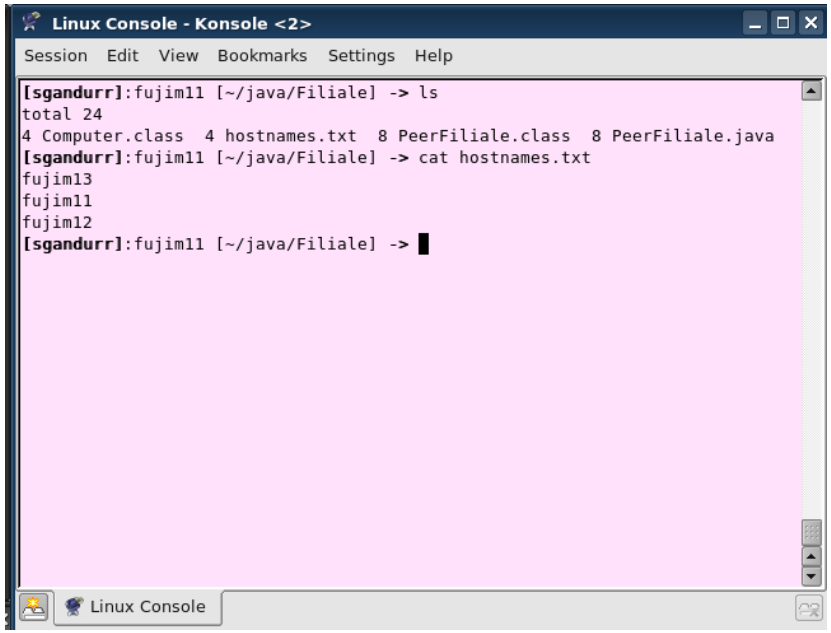


Secondo Giro: Comunicazione del Massimo



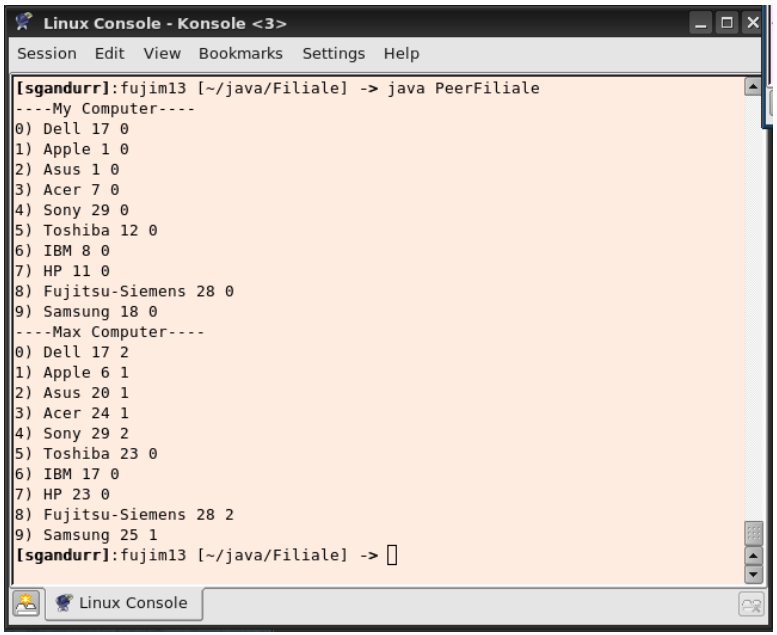
Connettersi agli Host

- Il file sorgente e' `PeerFiliale.java`. Compilarlo.
- Supponiamo che le PeerFiliali siano i tre host `fujim11`, `fujim12` e `fujim13`. Connettersi in ssh a tutti e tre gli host.
- Nella directory con il file class, deve essere presente il file `hostnames.txt` che contiene, uno per riga, tutti gli hostname dei PeerFiliale.

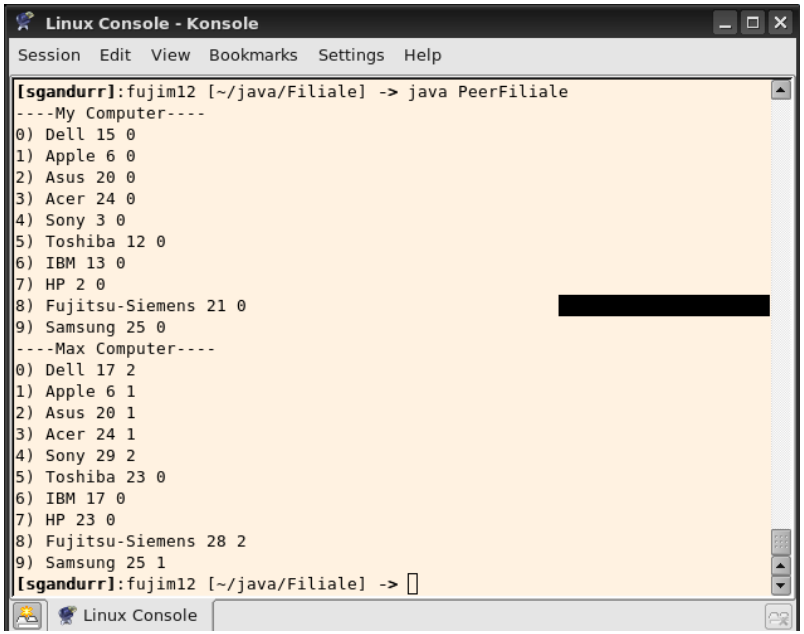


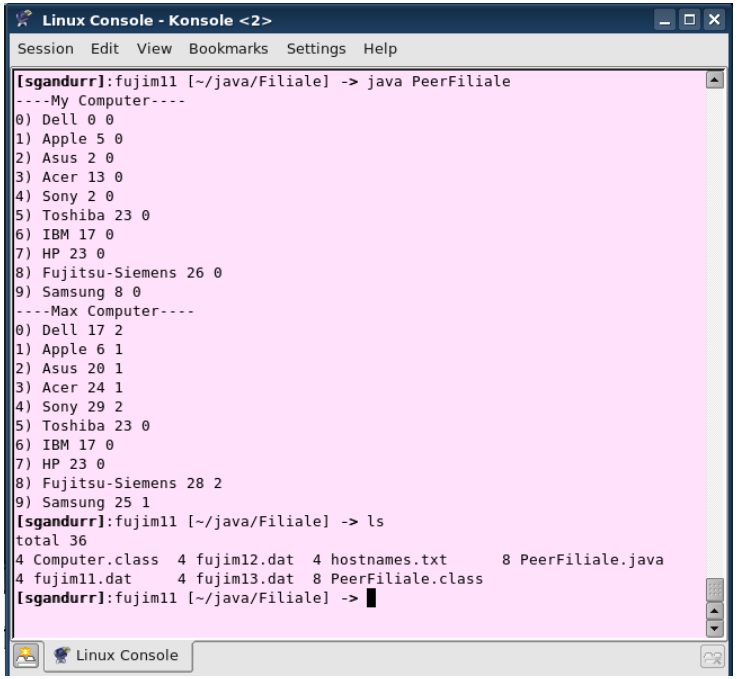
Avviare il Programma

- L'host con l'indice piu' basso deve essere avviato per ultimo (in questo caso `fujim11`).
- Il programma carica le giacenze da un file (`hostname.dat`): se non esiste ne crea uno con valori random.
- Per avviare il programma: `java PeerFiliale`



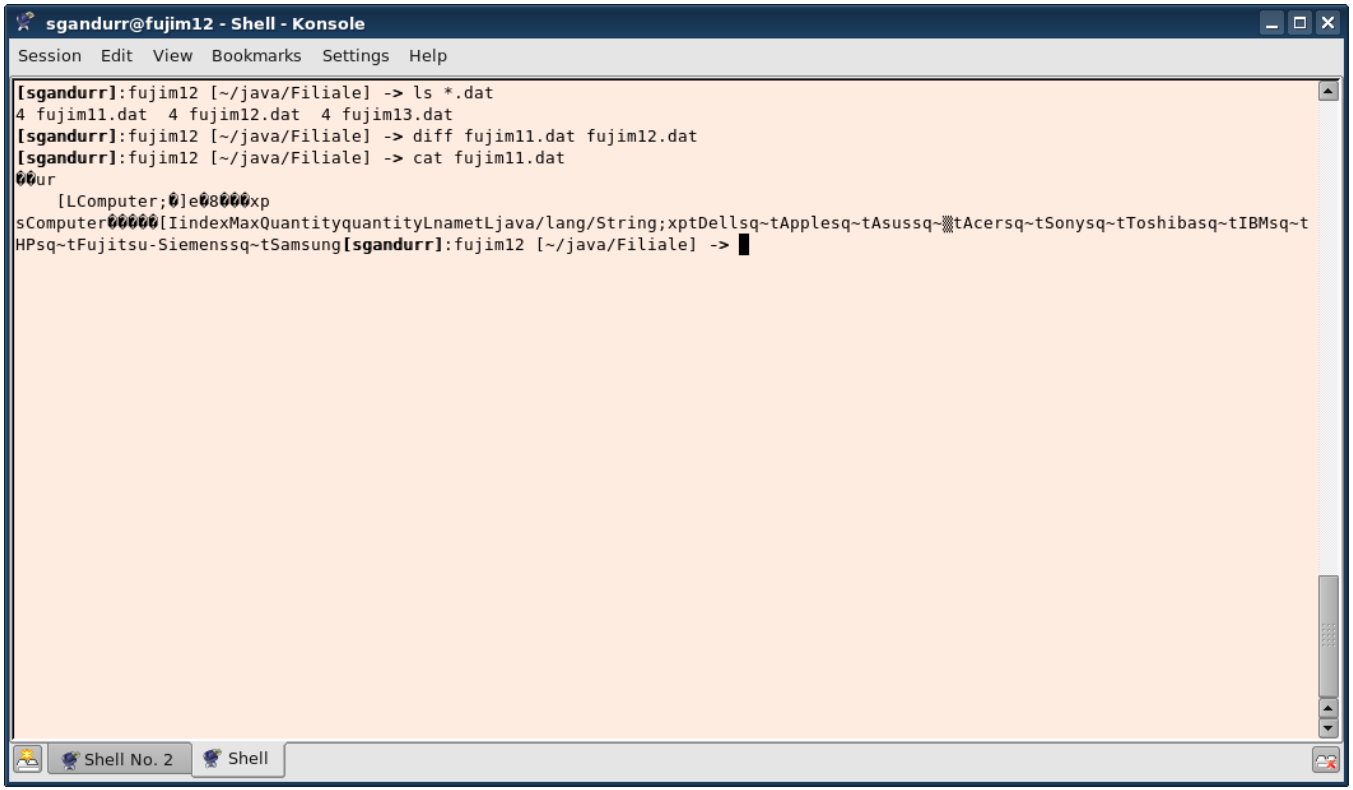
Avviare il Programma





I File .dat

Alla fine dell'esecuzione del programma, la directory conterra' un file `.dat` contenente gli oggetti Computer, uno per PeerFiliale.



(In questo caso tutti gli host condividono la stessa directory con l'applicazione: tutti i file `.dat` sono visibili da ogni host.)

Serializzazione

Per poter memorizzare su file l'oggetto Computer, la classe deve implementare Serializable:

```
class Computer implements Serializable
{
   private static final long serialVersionUID = -7931133918385176485L;
   private String name;
   private int quantity;
   private int indexMaxQuantity;
   ...
}
```

Il campo `serialVersionUID` e' un identificativo universale di versione per una classe `Serializable`. Durante la deserializzazione, questo numero e' utilizzato per assicurarsi che la classe caricata corrisponda esattamente all'oggetto serializzato. Altrimenti, viene lanciata una `InvalidClassException`.

Scrittura su File

Per scrivere un oggetto Computer su file:

```
private Computer myComputer[] = new Computer[MAX_COMPUTERS];
....
try
{
   ObjectOutputStream out = new ObjectOutputStream(
            new FileOutputStream("nome_del_file.dat"));
   out.writeObject(myComputer);
   out.close();
} catch (IOException e)
{System.out.println(e);}
```

Con i metodi `writeObject/readObject` si possono scrivere/leggere solo oggetti (anche stringhe e array). I tipi primitivi invece hanno metodi ad hoc come `writeInt/readInt`, `writeDouble/readDouble` etc.

Lettura da File

Per leggere un oggetto Computer da file:

```
private Computer myComputer[];
....
try
{
   ObjectInputStream in = new ObjectInputStream(
         new FileInputStream("nome_del_file.dat"));
   this.myComputer = (Computer[]) in.readObject();
   in.close();
 } catch (Exception e){System.out.println(e);}
```

L'ordine con cui si leggono gli oggetti da file e il loro tipo e' importante. Ogni chiamata a `readObject` legge un altro oggetto di tipo `Object` che va quindi castato.

Invio su Socket

Per inviare un oggetto Computer su socket:

```
private Computer myComputer[] = new Computer[MAX_COMPUTERS];
....
DatagramSocket socket = new DatagramSocket(listeningPort);
ByteArrayOutputStream bout = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bout);
out.writeObject(myComputer[i]);
out.flush();
byte data[] = bout.toByteArray();
DatagramPacket outgoingPacket = new DatagramPacket(data,
                    data.length, peerHost, listeningPort);
socket.send(outgoingPacket);
bout.reset();
```

Ricezione da Socket

Per ricevere un oggetto Computer da un socket:

```
DatagramSocket socket = new DatagramSocket(listeningPort);
byte[] buffer = new byte[MAX_LENGTH];
DatagramPacket incomingPacket = new DatagramPacket(buffer,
                                                   buffer.length);
socket.receive(incomingPacket);
incomingPacket.setLength(buffer.length);
ObjectInputStream ois = new ObjectInputStream(
                new ByteArrayInputStream(incomingPacket.getData()));
Computer currentMax = (Computer) ois.readObject();
```

Socket per Invio e Ricezione

Lo stesso socket puo' essere utilizzato per ricevere ed inviare dati:

```
//RICEZIONE
DatagramSocket socket = new DatagramSocket(listeningPort);
byte[] buffer = new byte[MAX_LENGTH];
DatagramPacket incomingPacket = new DatagramPacket(buffer,
                                                   buffer.length);
socket.receive(incomingPacket);
...
//INVIO
ByteArrayOutputStream bout = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bout);
out.writeObject(currentMax);
out.flush();
byte data[] = bout.toByteArray();
DatagramPacket outgoingPacket = new DatagramPacket(data,
                    data.length, peerHost, listeningPort);
socket.send(outgoingPacket);
bout.reset();
```

Che Cos'e' un URL

URL e' un acronimo per Uniform Resource Locator:

- Un riferimento (un indirizzo) per una risorsa su Internet.
- Di solito un URL e' il nome di un file su un host sul World Wide Web.
- Ma puo' anche puntare ad altre risorse:
  - Una query per un database.
  - L'output di un comando.
- Ad es: `http://java.sun.com`
  1. `http`: identificativo del protocollo.
  2. `java.sun.com`: nome della risorsa.

Resource Name

Il nome di una risorsa e' composto da:

- Host Name: il nome dell'host su cui si trova la risorsa.
- Filename: il pathname del file sull'host.
- Port Number: il numero della porta su cui connettersi (opzionale).
- Reference: un riferimento ad una specifica locazione all'interno del file (opzionale).

Nel caso del protocollo `http`, se il Filename e' omesso (o finisce per `/`), il server web e' configurato per restituire un file di default all'interno del path (ad es. `index.html`, `index.php`, `index.asp`).

URL e URI

- Un URI (Uniform Resource Identifier) e' un costrutto sintattico che specifica, tramite le varie parti che lo compongono, una risorsa su Internet.
  - `[schema:]ParteSpecificaSchema[#frammento]`
  - `ParteSpecificaSchema` ha la struttura: `[//autorita']][percorso][?query]`
- Un URL e' un tipo particolare di URI: quella parte che contiene sufficienti informazioni per individuare una risorsa.
- Altre URI, ad es: `mailto:user@hostname.com` non specificano come individuare la risorsa. In questo caso, le URI sono dette URN (Uniform Resource Name).

URL in JAVA

- In JAVA per creare un oggetto URL:

  ```
  URL cli = new URL("http://www.cli.di.unipi.it/");
  ```

  che e' un esempio di un URL assoluto.
- E' anche possible creare un URL relativo, che ha la forma *URL(URL baseURL, String relativeURL)*, ad es.

  ```
  URL cli = new URL("http://www.cli.di.unipi.it/");
  URL faq = new URL(cli, "faq.php");
  ```

  che risultera' puntare a `http://www.cli.di.unipi.it/faq.php`
- I protocolli gestiti da Java con gli URL sono `http`, `https`, `ftp`, `file` e `jar`.
- I costruttori di URL possono lanciare una MalformedURLException.

Parsare un URL

```
import java.net.*;
import java.io.*;

public class URLReader
{
   public static void main(String[] args) throws Exception
   {
      String url = "http://www.cli.di.unipi.it:80/faq.php?item=389";
      URL cli = new URL(url);

      System.out.println("protocol = " + cli.getProtocol());
      System.out.println("authority = " + cli.getAuthority());
      System.out.println("host = " + cli.getHost());
      System.out.println("port = " + cli.getPort());
      System.out.println("path = " + cli.getPath());
      System.out.println("query = " + cli.getQuery());
      System.out.println("filename = " + cli.getFile());
      System.out.println("ref = " + cli.getRef());
   }
}
```

Parsare un URL

Nell'esempio di prima, usando come URL:
`http://www.cli.di.unipi.it:80/faq.php?item=389` si ottiene:

```
protocol = http
authority = www.cli.di.unipi.it:80
host = www.cli.di.unipi.it
port = 80
path = /faq.php
query = item=389
filename = /faq.php?item=389
ref = null
```

Leggere un URL

Una volta creato un oggetto URL si puo' invocare il metodo `openStream()` per ottenere uno stream da cui poter leggere il contenuto dell'URL.

- Il metodo `openStream()` ritorna un oggetto `java.io.InputStream`: leggere da un URL e' analogo a leggere da uno stream di input.

```
import java.net.*;
import java.io.*;
public class URLReader
{  public static void main(String[] args) throws Exception
   {
      String url = "http://www.cli.di.unipi.it/";
      URL cli = new URL(url);
      BufferedReader in = new BufferedReader(
         new InputStreamReader(cli.openStream()));
      String inputLine;
      while((inputLine = in.readLine()) != null)
         System.out.println(inputLine);
      in.close();
    }
}
```

Leggere un URL

Nell'esempio di prima, leggendo l'URL `http://www.cli.di.unipi.it/`, si ottiene:

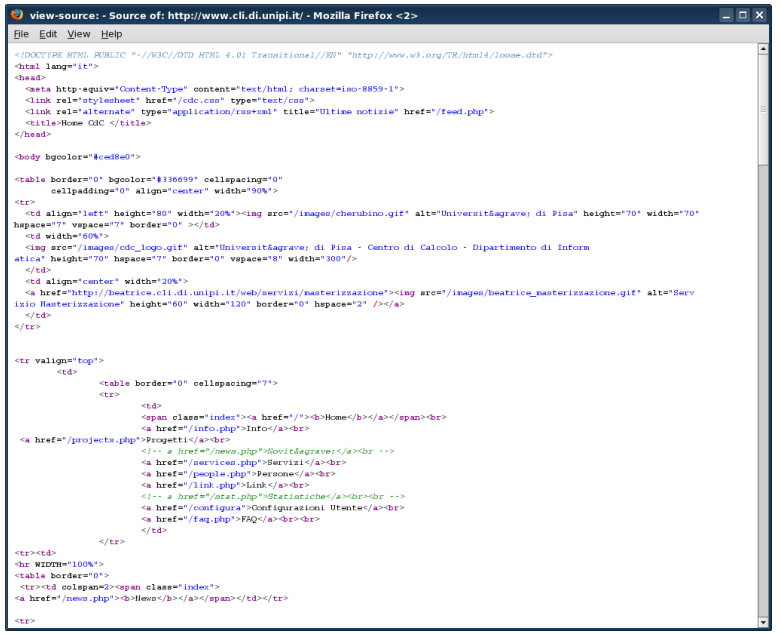
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="it">
<head>
  <meta http-equiv="Content-Type" content="text/html;
                                   charset=iso-8859-1">
  <link rel="stylesheet" href="/cdc.css" type="text/css">
  <link rel="alternate" type="application/rss+xml"
                 title="Ultime notizie" href="/feed.php">
  <title>Home CdC </title>
</head>
<body bgcolor="#ced8e0">
....
```

Se necessario, settare il proxy con:

```
java -Dhttp.proxyHost=proxyhost
    [-Dhttp.proxyPort=portNumber] URLReader
```

Page Source

Confrontare l'output dell'esempio precedente con il page source del browser:



Connettersi ad un URL

- Nell'esempio precedente, la connessione all'URL veniva effettuata solo dopo aver invocato `openStream`.
- In alternativa, e' possibile invocare il metodo `openConnection` per ottenere un oggetto `URLConnection`.
- Utile nel caso in cui si vogliano settare alcuni parametri o proprieta' della richiesta prima di connettersi.
  - es:
    `cliConn.setRequestProperty(``User-Agent'', ``Mozilla/5.0'');`
- Successivamente, per connettersi deve essere invocato il metodo `URLConnection.connect`.

```
...
String url = "http://www.cli.di.unipi.it/";
URL cli = new URL(url);
URLConnection cliConn = cli.openConnection();
cliConn.connect();
BufferedReader in = new BufferedReader(
      new InputStreamReader(
      cliConn.getInputStream()));
...
```