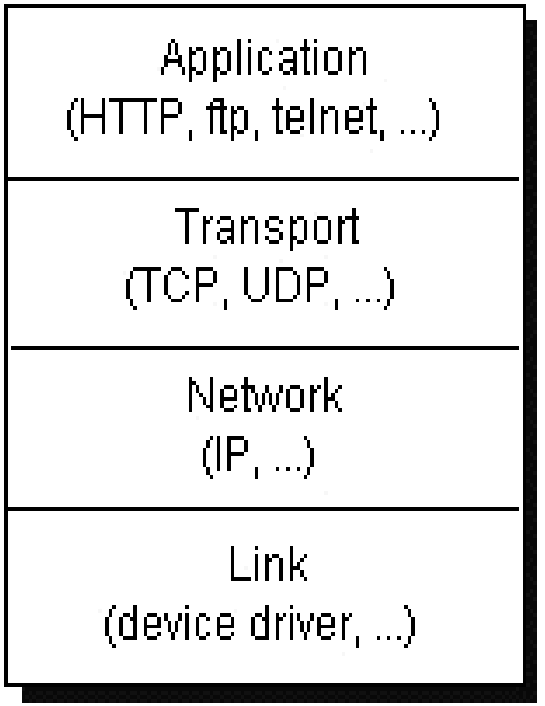
Programare avansată

Programare în rețea

Protocol

- *Protocol* = Mulțime de reguli (convenții) care permit interacțiunea între doi sau mai mulți parteneri



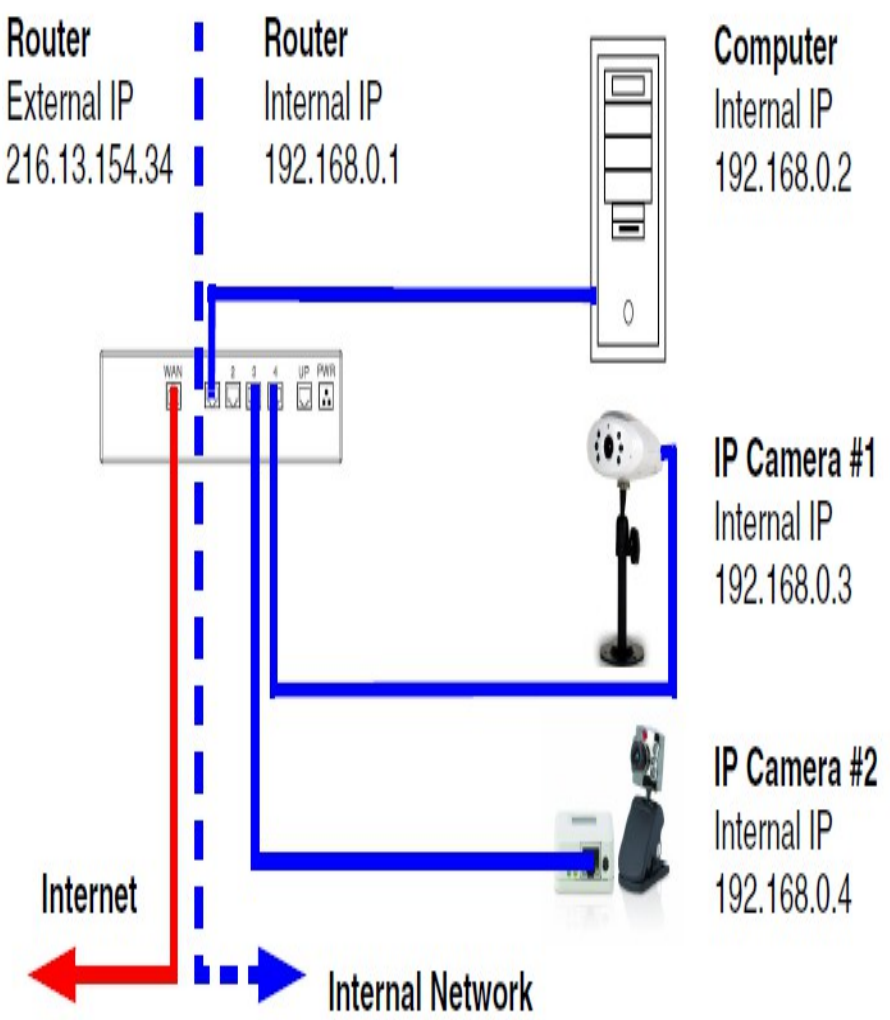
Adresă IP

java.net.InnetAddress

- *Innet4Address* (32-bit)
  - ✔ 85.122.23.145 - numerică
  - ✔ fenrir.info.uaic.ro - simbolică
- *Innet6Address* (128-bit)
  2002:0:0:0:0:0:557a:1791

Tipuri de adrese

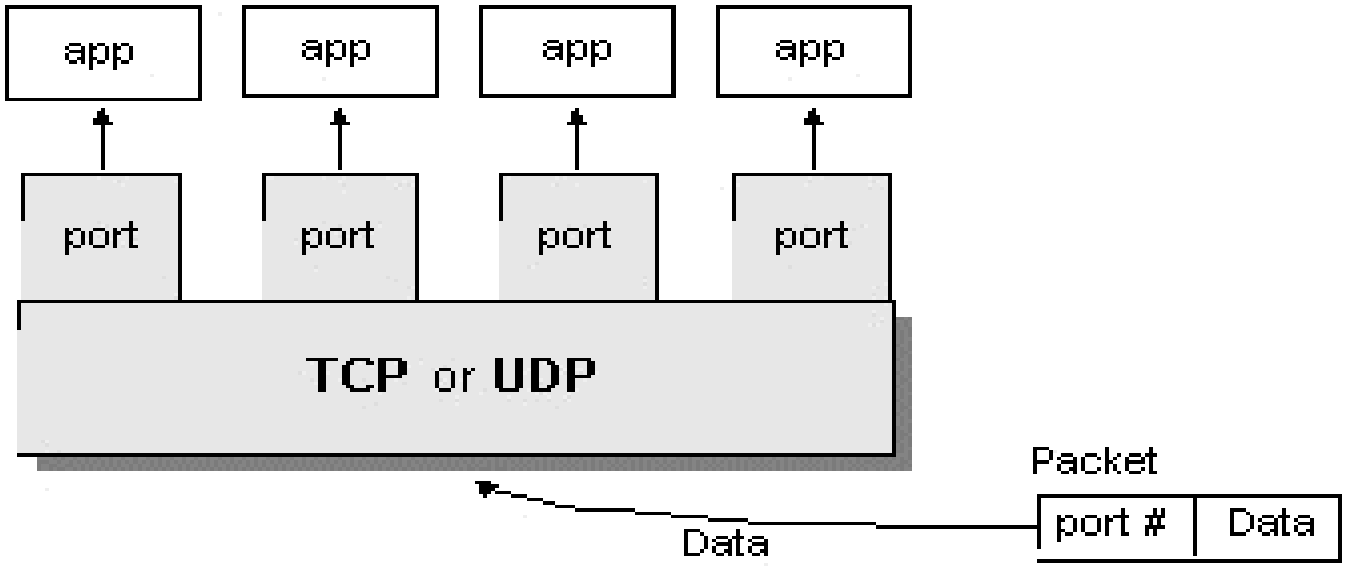
- ◆ unicast
- ◆ multicast
- ◆ localhost (127.0.0.1)



Port

Un *port* este un număr pe 16 biți care identifică în mod unic un proces care oferă servicii în rețea.

- Valori posibile: 0 − 65535
- Valori rezervate: 0 − 1023



Modelul *client-server*

- **Serverul**
  - oferă servicii în rețea
  - are un *port* atașat
  - trebuie să poată trata cereri concurente

- **Clientul**
  - inițiază conversația cu server-ul
  - trebuie să cunoască adresa IP și portul serverului
  - solicită un anumit serviciu

Socket-uri

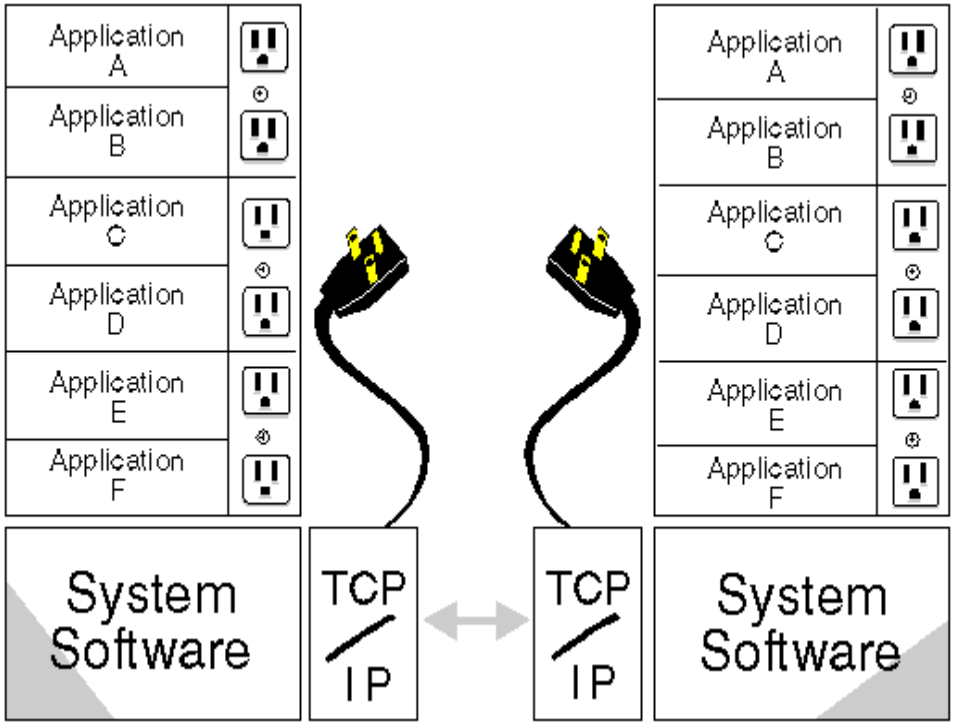
Socket (soclu) = abstracțiune software care descrie ”capetele” unei conexiuni

- TCP: *Socket, ServerSocket*
- UDP: *DatagramSocket*

java.net.InetSocketAddress

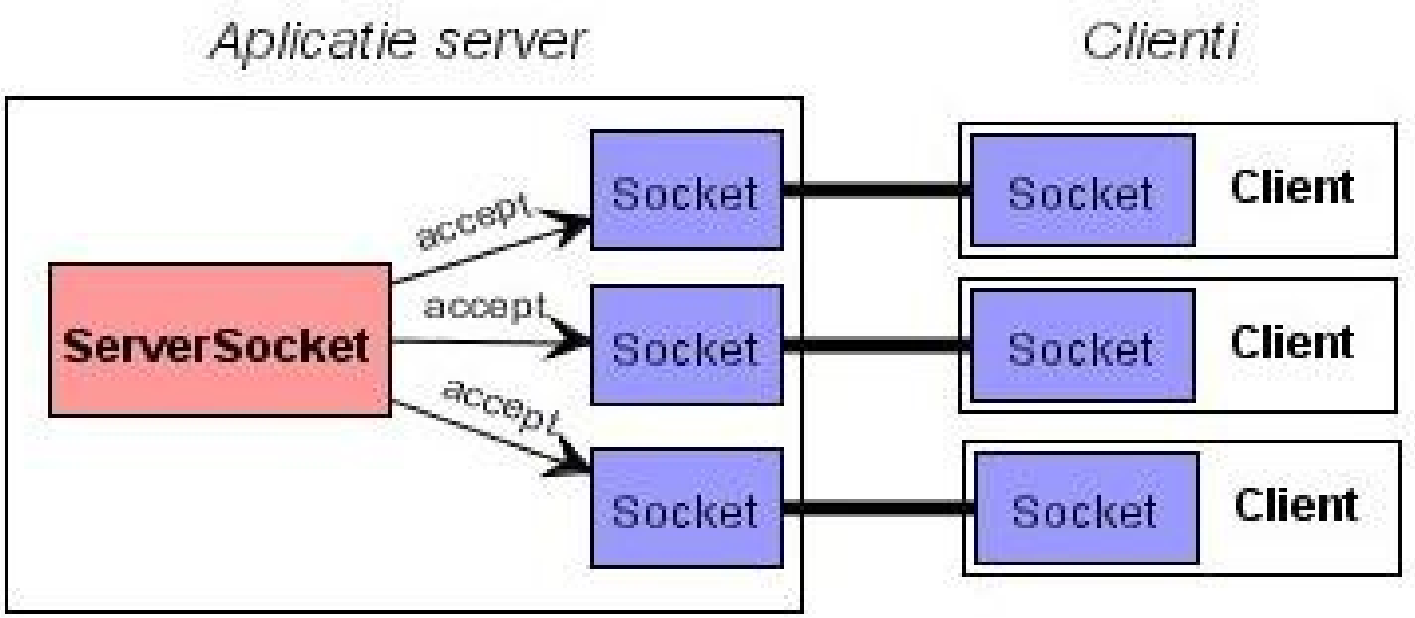
(IP adress, port)

(hostName, port)



Comunicarea folosind TCP

- *Transport Control Protocol*
- Flux sigur de date
- Conexiune permanentă pe parcursul comunicației



Structura unui server TCP

```
public class SimpleServer {
  // Definim portul la care se ruleaza serverul
  public static final int PORT = 8100;

  public SimpleServer() throws IOException {
    ServerSocket serverSocket = null ;
    try {
      serverSocket = new ServerSocket( PORT );
      while ( true ) {
        System.out.println (" Asteptam un client ...");
        Socket socket = serverSocket.accept();
        // Executam solicitarea clientului intr -un fir de executie
        new ClientThread(socket).start();
      }
    } catch ( IOException e) {
      System.err. println (" Eroare IO \n" + e);
    } finally {
      serverSocket.close();
    }
  }
  public static void main ( String [] args ) throws IOException {
    SimpleServer server = new SimpleServer ();
  }
}
```

Crearea răspunsului

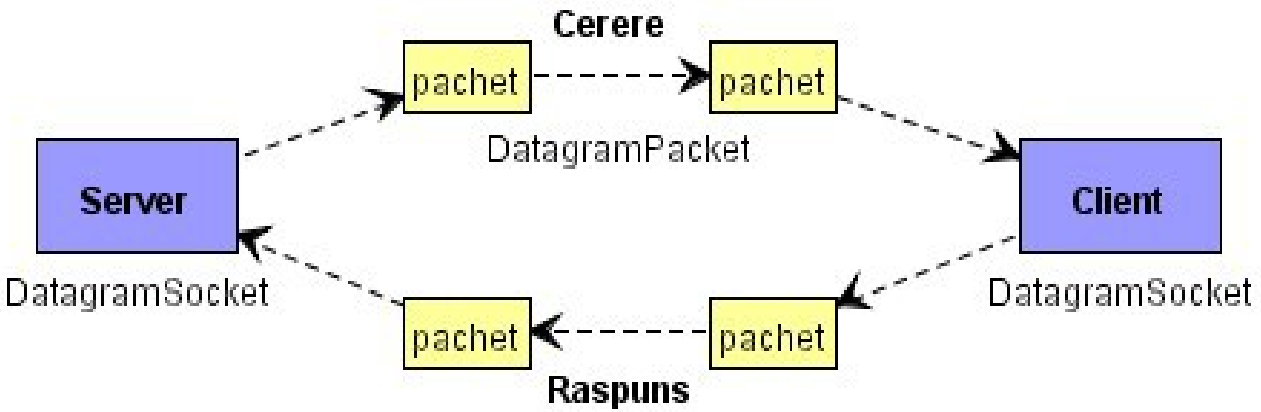
```
class ClientThread extends Thread {
  private Socket socket = null ;
  public ClientThread ( Socket socket ) { this.socket = socket ; }
  public void run () {
    try {
      // in este fluxul de intrare de la client
      BufferedReader in = new BufferedReader (
          new InputStreamReader (socket.getInputStream()));
      String cerere = in.readLine (); // Primim cerere de la client

      // out este flux de iesire catre client
      PrintWriter out = new PrintWriter (socket.getOutputStream());
      String raspuns = " Hello " + cerere + "!";
      out.println ( raspuns ); // Trimitem raspunsul clientului
      out.flush ();
    } catch (IOException e) {
      System.err.println("Eroare IO \n" + e);
    } finally { // Inchidem socketul deschis pentru clientul curent
      try {
        socket.close();
      } catch ( IOException e) { System.err. println (e); }
    }
  }
}
```

Structura unui client TCP

```
public class SimpleClient {
  public static void main (String[] args) throws IOException {
    String adresaServer = " 127.0.0.1 "; // Adresa IP a serverului
    int PORT = 8100; // Portul la care serverul ofera serviciul
    Socket socket = null ;
    PrintWriter out = null ;
    BufferedReader in = null ;
    try {
      socket = new Socket (adresaServer, PORT);
      // Trimitem o cerere la server
      out = new PrintWriter (socket.getOutputStream(), true);
      String cerere = "Duke";
      out.println (cerere);
      out.close();

      // Asteaptam raspunsul de la server ("Hello Duke !")
      in = new BufferedReader (
             new InputStreamReader (socket.getInputStream ()));
      raspuns = in.readLine ();
      in.close();
      System.out.println(raspuns);
    } catch ( UnknownHostException e) {
      System.err.println (e);
    } finally { socket.close(); }
}
```

Comunicarea folosind UDP

- *User Datagram Protocol*
- Pachete independente de date (datagrame)
- Nu stabilește o conexiune permanentă
- Ordonea sau chiar ajungerea pachetelor la destinație nu sunt garantate



Structura unui server UDP

```
int portServer = 8200; //portul serverului

// Construim un socket pentru comunicare
DatagramSocket socket = new DatagramSocket(portServer);

// Asteaptam pachetul cu cererea
byte buf[] = new byte[256];
DatgramPacket cerere = new DatagramPacket(buf, buf.length );
socket.receive(cerere);

// Aflam adresa si portul de la care vine cererea
InetAddress adresaClient = cerere.getAddress();
int portClient = cerere.getPort();

// Construim raspunsul
String mesaj = " Hello " + new String(cerere.getData());
buf = mesaj.getBytes();

// Trimitem un pachet cu raspunsul catre client
DatagramPacket raspuns =
    new DatagramPacket(buf, buf.length, adresaClient, portClient);
socket.send(raspuns);
```

Structura unui client UDP

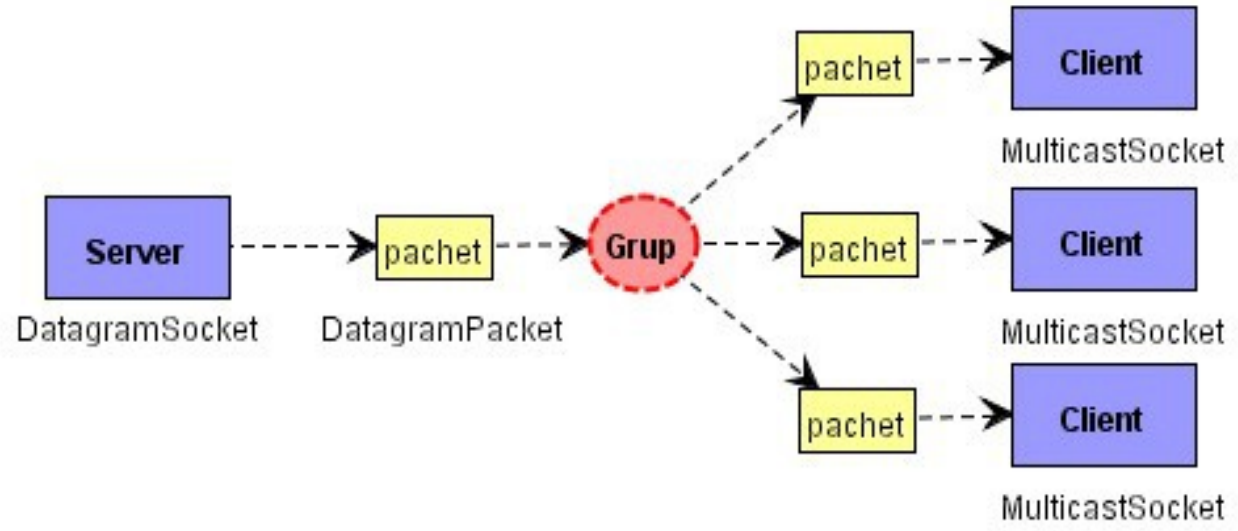
```
InetAddress adresaServer = InetAddress.getByName("127.0.0.1");
int portServer = 8200; //portul serverului

// Construim un socket pentru comunicare
// Portul acestui socket este ales automat dintre cele disponibile
DatagramSocket socket = new DatagramSocket();

// Construim si trimitem pachetul cu cererea
byte buf[] = "Duke".getBytes ();
DatagramPacket cerere =
  new DatagramPacket(buf, buf.length, adresaServer, portServer);
socket.send(cerere);

// Asteaptam pachetul cu raspunsul
buf[] = new byte [256];
DatagramPacket raspuns = new DatagramPacket(buf, buf.length );
socket.receive(raspuns);

// Afisam raspunsul
System.out.println(new String(raspuns.getData()));
```

Trimiterea de mesaje către un grup

Un grup de clienți abonați pentru operația de *multicast* este specificat printr-o adresă IP din intervalul:

224.0.0.1- 239.255.255.255

```
InetAddress group = InetAddress.getByName("230.0.0.1");
// Ne alaturam grupului aflat la adresa si portul specificate
MulticastSocket clientSocket = new MulticastSocket (4444);
clientSocket.joinGroup(group);
```



Comunicarea cu resurse Web

URL = Uniform Resource Locator

- resurse statice (pagini html, text, imagini, etc.)

`http://thor.info.uaic.ro/~acf/java/slides/prog_retea_slide.pdf`

- componente *sever-side* (servlet, pagina jsp, php etc.) ce generează conținut dinamic

`http://85.122.23.145:8080/WebApplication/hello.jsp`

Componentele unui URL:

- Identificatorul protocolului
- Identificatorul resursei referite
  - Adresa sau numele calculatorului gazdă, portul
  - Calea completă spre resursă

Lucrul cu URL-uri

java.net.URL

```
try {
  URL adresa = new URL("http://xyz.abc");
} catch (MalformedURLException e) {
  System.err.println("URL invalid: " + e);
}
```

- Aflarea informațiilor despre resursa referită
- Citirea conținutului
- Conectarea la acel URL

Citirea conținutului unei resurse Web

```
import java.net .*;
import java.io.*;
public class CitireURL {
  public static void main(String[] args) throws IOException {
    String adresa = " http://www.infoiasi.ro";
    BufferedReader br = null ;
    try {
      URL url = new URL (adresa);
      InputStream in = url.openStream ();
      br = new BufferedReader (new InputStreamReader (in));
      String linie ;
      while (( linie = br. readLine ()) != null ) {
        // Afisam linia citita
        System.out.println (linie);
      }
    } catch ( MalformedURLException e) {
      System . err. println ("URL invalid: " + e);
    } finally {
      if (br != null) br. close ();
    }
  }
}
```

Conectarea la un URL

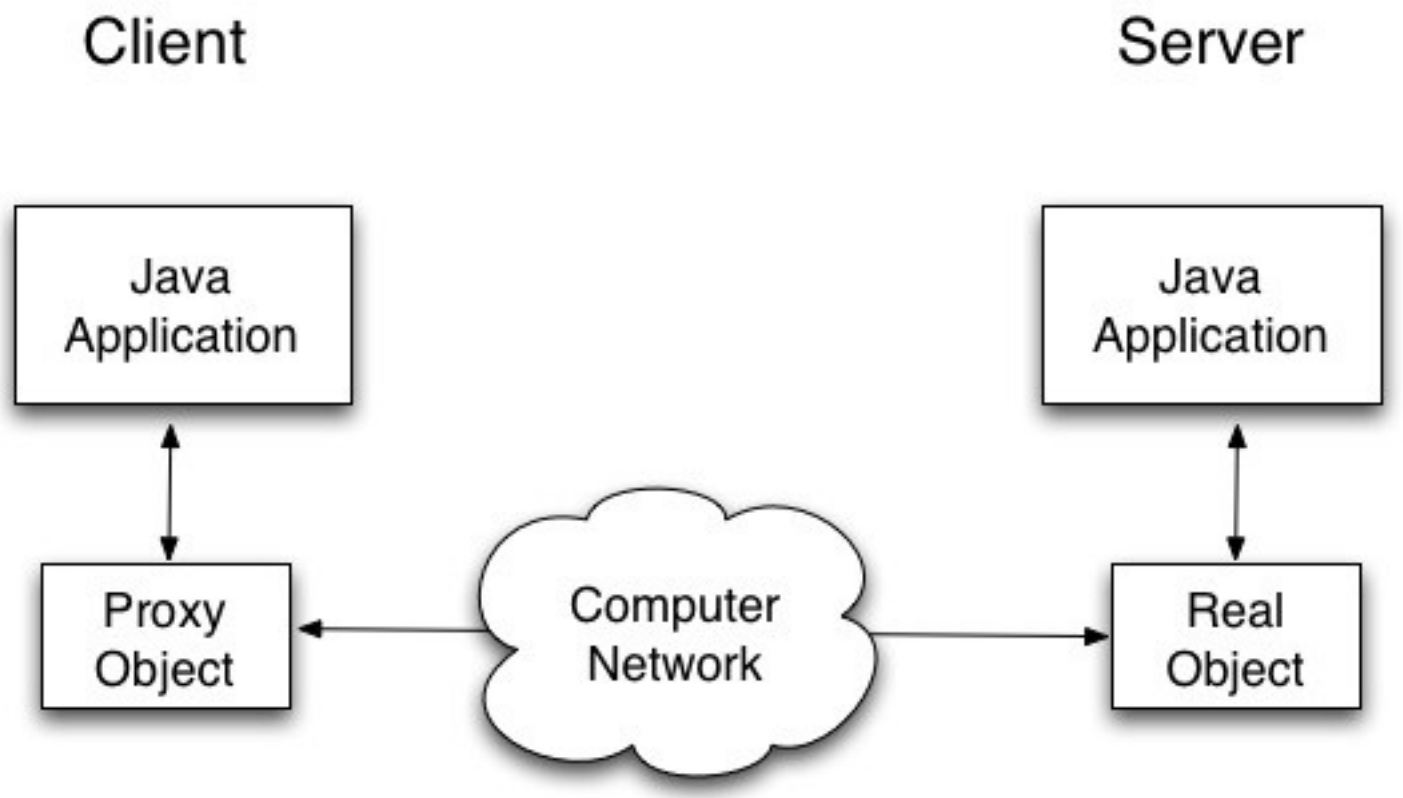
```
import java.net .*;
import java.io.*;
public class ConectareURL {
  public static void main(String[] args) throws IOException {
    URL url = new URL("http://localhost:8080/App/HelloWorld);
    URLConnection connection = url.openConnection();
    connection.setDoOutput(true);

    OutputStreamWriter out =
        new OutputStreamWriter(connection.getOutputStream());
    String param = URLEncoder.encode("Duke & World", "UTF-8");
    out.write("string=" + param);
    out.close();

    BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
    String response;
    while ((response = in.readLine()) != null) {
      System.out.println(response);
    }
    in.close();
  }
}
```

Remote Method Invocation (RMI)

- Programare de rețea la un nivel superior
- Implementarea facilă a **aplicațiilor distribuite**
- Colaborarea obiectelor aflate în mașini virtuale diferite; o aplicație poate să apeleze metode ale unui obiect aflat în alt spațiu de adrese:
  - ✔ identificarea obiectelor externe (*remote*)
  - ✔ trimiterea parametrilor și primirea rezultatelor
  - ✔ tratarea excepților și gestiunea memoriei
- Sintaxă și semantică similare cu cele ale aplicațiilor standard (ne-distribuite)

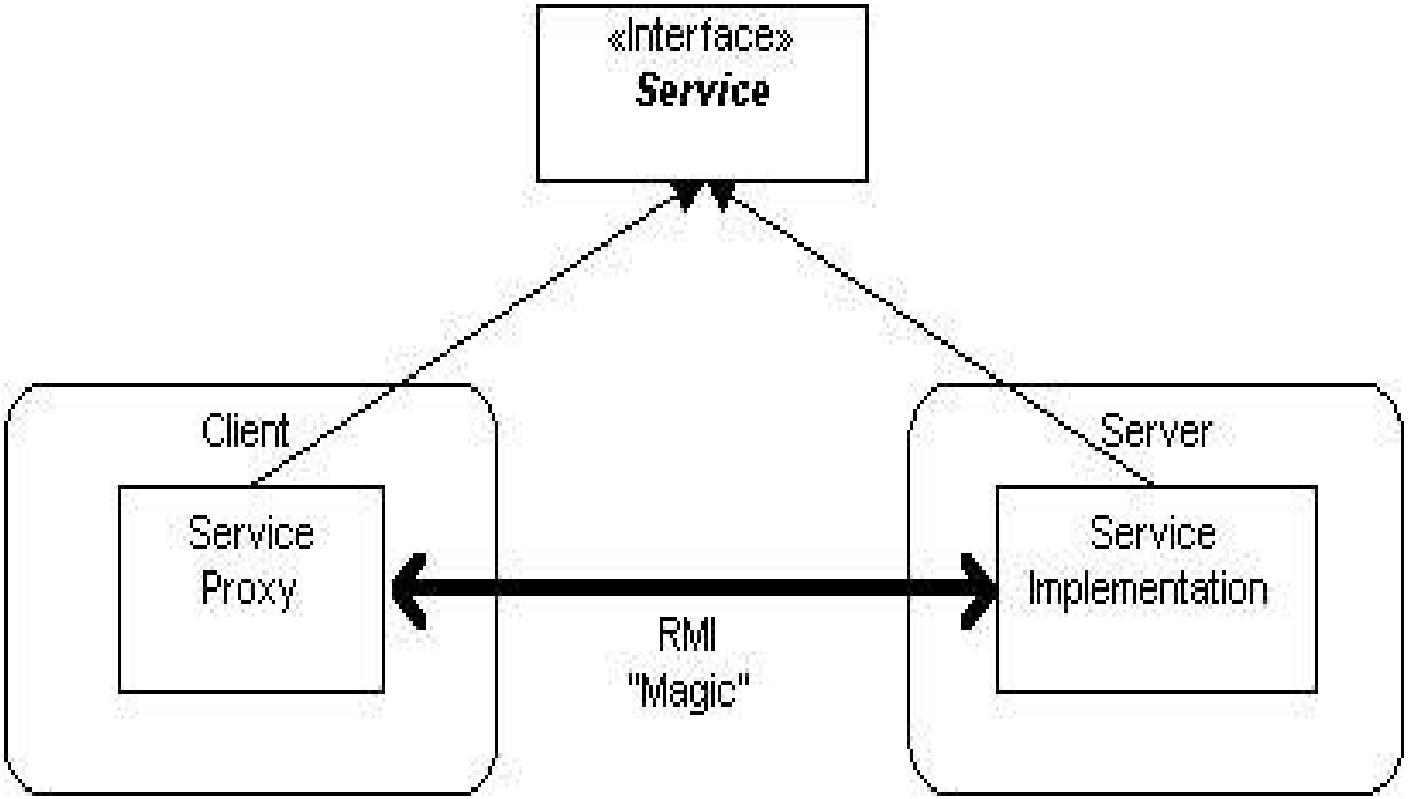
Remote Proxy

Proxy = obiect care funcționează ca o interfață către alt obiect (din rețea, memorie, etc.)



Principiul de bază

Separarea conceptelor de comportament și implementare



Identificarea obiectelor în rețea

Servicii de nume

- **JNDI** (Java Naming and Directory Interface)
- **RMI Registry**

Operații uzuale ale serviciilor de nume

- ***bind*** - asocierea dintre un obiect și un nume simbolic
- ***lookup*** - obținerea referinței unui obiect pe baza numelui

Exemplu de utilizare RMI

Interfața Hello.java – reprezintă descrierea serviciului

Va fi disponibilă atât serverului cât și clientului.

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

public interface Hello extends Remote {
  String sayHello(String name) throws RemoteException;
}
```

Implementare serviciului

Clasa HelloImpl.java – se găsește doar la nivelul serverului

```
package server;
import java.rmi.RemoteException;
import service.Hello;
public class HelloImpl implements Hello {
  public HelloImpl() throws RemoteException {
    super();
  }
  public String sayHello(String name) {
    return "Hello " + name;
  }
}
```

Expunerea serviciului în rețea

Clasa HelloServer.java

```
package server;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import service.Hello;
public class HelloServer {
  public static void main(String[] args) throws Exception {
    Hello hello = new HelloImpl();
    Hello stub = (Hello) UnicastRemoteObject.exportObject(hello, 0);
    Registry registry = LocateRegistry.getRegistry();
    registry.bind("Hello", stub);
    System.out.println("Serviciul Hello activat!");
  }
}
```

Aplicația client

Clasa HelloClient.java

```
package client;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import service.Hello;

public class HelloClient{

  public static void main(String[] args) throws Exception {

    Registry registry = LocateRegistry.getRegistry("localhost");

    Hello hello = (Hello) registry.lookup("Hello");

    String response = hello.sayHello("Duke");

    System.out.println(response);

  }

}
```