# Programare în reţea

- **Introducere**
- **Lucrul cu URL-uri**
- **Socket-uri**
- **Comunicarea prin conexiuni**
- **Comunicarea prin datagrame**
- **Trimiterea de mesaje către mai mulţi clienţi**
- **Tehnologii de programare in reţea**

## Introducere

Trimiterea de mesaje şi date între aplicaţii aflate pe calculatoare conectate printr-o reţea.

Pachetul care oferă suport pentru scrierea aplicaţiilor de reţea este **java.net**

Noţiuni fundamentale referitoare la reţele:

- **protocol**
- **adresa IP**
- **port**
- **socket**

## Ce este un protocol ?

*Protocol* = convenţie de reprezentare a datelor

- **TCP (Transport Control Protocol)**
  - flux sigur de date
  - conexiune permanentă pe parcursul comunicaţiei
- **UDP (User Datagram Protocol)**
  - pachete independente de date, numite *datagrame*
  - nu stabileşte o conexiune permanentă

– ajungerea pachetelor la destinaţie nu este garantată

## Cum este identificat un calculator în reţea ?

**Adresa IP** (IP - *Internet Protocol*)

- **numerică**
  număr pe 32 de biţi
  4 octeţi: `193.231.30.131`
- **simbolica** : `thor.infoiasi.ro`

Clasa **InetAddress**

**Numele calculatorului**
Permite identificarea într-o reţea locală.

## Ce este un port ?

Un *port* este un număr pe 16 biţi care identifică în mod unic procesele care rulează pe o anumită masină.

Orice aplicaţie care realizează o conexiune în reţea va trebui să ataşeze un număr de port acelei conexiuni.

Valori posibile: $0 - 65535$

Valori rezervate: $0 - 1023$

# Clase de bază din java.net

| TCP | UDP |
|---|---|
| URL | DatagramPacket |
| URLConnection | DatagramSocket |
| Socket | MulticastSocket |
| ServerSocket | |

# Lucrul cu URL-uri

## Uniform Resource Locator

- fişier (pagină Web, text, imagine, etc.)
- interogări la baze de date
- rezultate ale unor comenzi

```
http://java.sun.com
http://students.infoiasi.ro/index.html
http://www.infoiasi.ro/~acf/imgs/taz.gif
http://www.infoiasi.ro/~acf/
  java/curs/9/prog_retea.html#url
```

Componentele unui URL:

- Identificatorul protocolului
- Numele resursei referite
  - Numele calculatorului gazdă
  - Calea completă spre resursa
  - O referinţă de tip *anchor*
  - Un port

## Clasa java.net.URL

Permite crearea unei referinţe la un anumit URL.

Poate genera **MalformedURLException**

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

- Aflarea informaţiilor despre resursa referită
- Citirea printr-un flux a continutului fişierului respectiv.
- Conectarea la acel URL pentru citirea şi scrierea de informaţii.

# Citirea conţinutului unui URL

Listing 1: Citirea conţinutului unui URL

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

public class CitireURL {
  public static void main(String[] args)
      throws IOException{
    String adresa = "http://www.infoiasi.ro";
    if (args.length > 0)
      adresa = args[0];

    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 !\n" + e);
    } finally {
      br.close();
    }
  }
}
```

## Conectarea la un URL

**openConnection** - crează o conexiune bidirecţională cu resursa specificată.

Conexiunea este un obiect de tip **URLConnection** şi permite:

- crearea unui flux de intrare pentru citire
- creare unui flux de ieşire pentru scrierea de date
  URL: jsp, servlet, cgi-bin, php, etc

## Socket-uri

*Socket (soclu)* = abstracţiune software; "capetele" unei conexiuni; ataşat unui port.

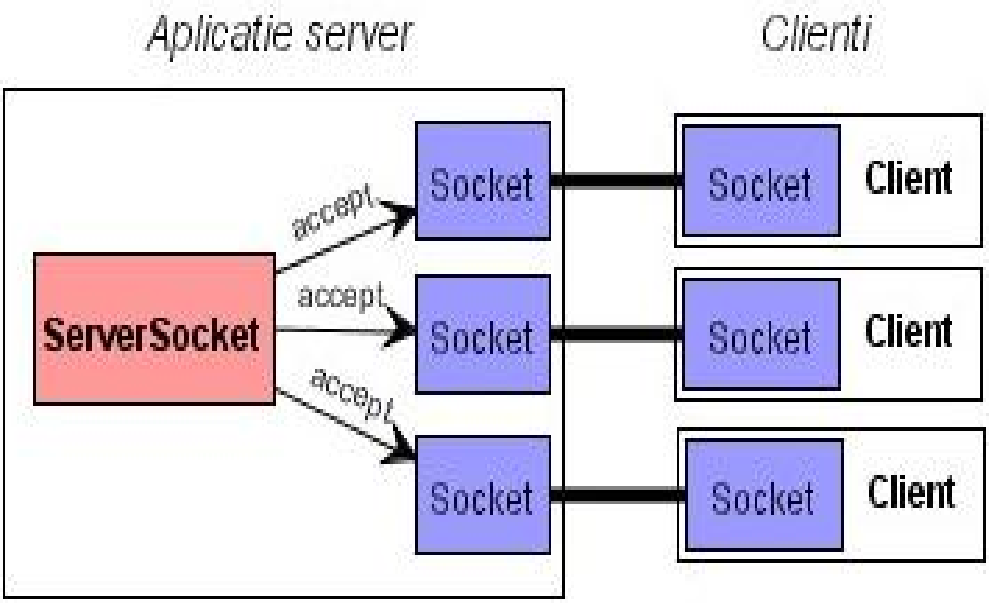
- TCP: `Socket` şi `ServerSocket`
- UDP: `DatagramSocket`

## Modelul client/server

- serverul - oferă servicii; trebuie să poată trata cereri concurente.
- clientul - iniţiază conversaţia cu un server, solicitând un anumit serviciu.

**InetSocketAddress** - (adresa IP, număr port)

# Comunicarea prin conexiuni




```
ServerSocket(int port)
  accept
Socket(InetAddress address, int port)
  getInputStream
  getOutputStream
```

- `BufferedReader`, `BufferedWriter` şi `PrintWriter`
- `DataInputStream`, `DataOutputStream`
- `ObjectInputStream`, `ObjectOutputStream`

# Structură server

```
1. Creeaza un obiect de tip ServerSocket
   la un anumit port
while (true) {
  2. Asteapta realizarea unei conexiuni cu un client,
     folosind metoda accept;
     (va fi creat un obiect nou de tip Socket)
  3. Trateaza cererea venita de la client:
     3.1 Deschide un flux de intrare si
         primeste cererea
     3.2 Deschide un flux de iesire si
         trimite raspunsul
     3.3 Inchide fluxurile si socketul nou creat
}
```

# Structură client

```
1. Citeste sau declara adresa IP a serverului si
   portul la care acesta ruleaza;
2. Creeaza un obiect de tip Socket
   cu adresa si portul specificate;
3. Comunica cu serverul:
   3.1 Deschide un flux de iesire si trimite cererea;
   3.2 Deschide un flux de intrare si primeste
       raspunsul;
   3.3 Inchide fluxurile si socketul creat;
```

Listing 2: Structura unui server bazat pe conexiuni

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

class ClientThread extends Thread {
  Socket socket = null;

  public ClientThread(Socket socket) {
    this.socket = socket;
  }

  public void run() {
    //Executam solicitarea clientului
    String cerere, raspuns;
    try {
      // in este fluxul de intrare de la client
      BufferedReader in = new BufferedReader(new
          InputStreamReader(
        socket.getInputStream() ));

      // out este flux de iesire catre client
      PrintWriter out = new PrintWriter(
        socket.getOutputStream());

      // Primim cerere de la client
      cerere = in.readLine();

      // Trimitem raspuns clientului
      raspuns = "Hello " + cerere + "!";
      out.println(raspuns);
      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("Socketul nu poate fi inchis \n" +
            e);
      }
    }
  }
```

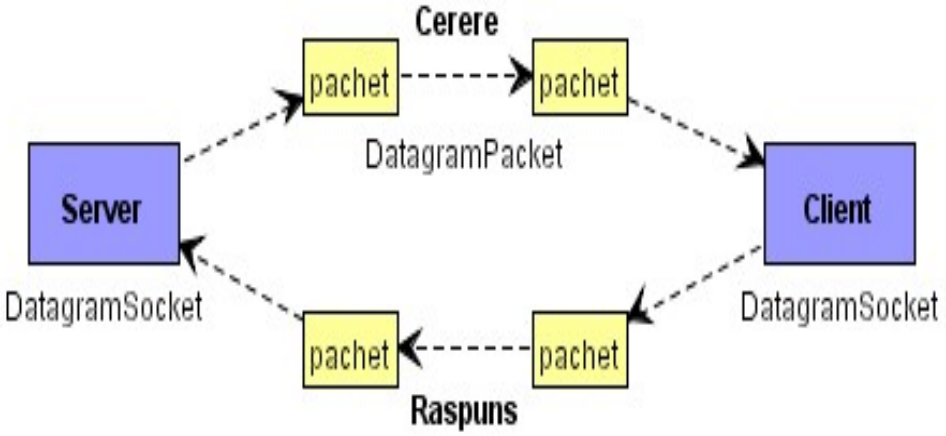
```
}

public class SimpleServer {

  // Definim portul pe care se gaseste serverul
  // (in afara intervalului 1-1024)
  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
        ClientThread t = new ClientThread(socket);
        t.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();
  }
}
```

Listing 3: Structura unui client bazat pe conexiuni

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

public class SimpleClient {

  public static void main(String[] args) throws IOException {
    // Adresa IP a serverului
    String adresaServer = "127.0.0.1";
```

```
    // Portul la care serverul ofera serviciul
    int PORT = 8100;

    Socket socket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    String cerere, raspuns;

    try {
      socket = new Socket(adresaServer, PORT);
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(
         socket.getInputStream()));

      // Trimitem o cerere la server
      cerere = "Duke";
      out.println(cerere);

      //Asteaptam raspunsul de la server ("Hello Duke!")
      raspuns = in.readLine();
      System.out.println(raspuns);

    } catch (UnknownHostException e) {
      System.err.println("Serverul nu poate fi gasit \n" + e)
         ;
      System.exit(1);
    } finally {
      if (out != null)
        out.close();
      if (in != null)
        in.close();
      if (socket!= null)
        socket.close();
    }
  }
}
```

# Comunicarea prin datagrame




**Avantaj:** Solicită mai puţin serverul.

## Dezavantaje

- Nu este garantată ajungerea pachetelor la destinaţie
- Ordinea expedierii poate să nu fie păstrată

## Crearea unui pachet

```
//Epediere
DatagramPacket(byte[] buf, int length,
  InetAddress address, int port)
DatagramPacket(byte[] buf, int offset, int length,
  InetAddress address, int port)

DatagramPacket(byte[] buf, int offset, int length,
  SocketAddress address)
DatagramPacket(byte[] buf, int length,
  SocketAddress address)

//Receptionare
DatagramPacket(byte[] buf, int length)
DatagramPacket(byte[] buf, int offset, int length)
```

## Expedierea / Receptionarea

Metodele **send / receive** din clasa `DatagramSocket`

## Setarea informaţiilor

- **setData**
- `setAdress, setPort`
- `setSocketAddress`

## Extragerea informaţiilor

- **getData**
- `getAdress, getPort`
- `getSocketAddress`

Listing 4: Structura unui server bazat pe datagrame

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

public class DatagramServer {

  public static final int PORT = 8200;
  private DatagramSocket socket = null;
  DatagramPacket cerere, raspuns = null;

  public void start() throws IOException {

    socket = new DatagramSocket(PORT);
    try {
      while (true) {

        // Declaram pachetul in care va fi receptionata
           cererea
        byte[] buf = new byte[256];
        cerere = new DatagramPacket(buf, buf.length);

        System.out.println("Asteptam un pachet...");
        socket.receive(cerere);

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

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

        // Trimitem un pachet cu raspunsul catre client
        raspuns = new DatagramPacket(buf, buf.length, adresa,
            port);
        socket.send(raspuns);
      }
    } finally {
      if (socket != null)
        socket.close();
    }
  }
```

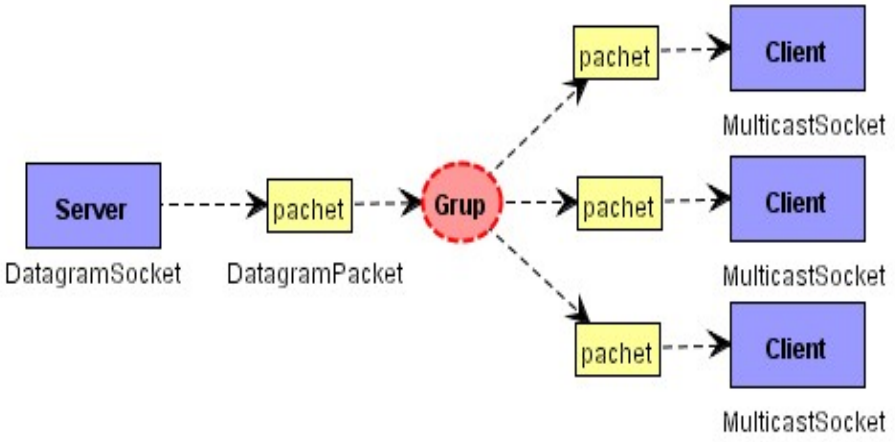
```
  public static void main(String[] args) throws IOException {
    new DatagramServer().start();
  }
}
```

Listing 5: Structura unui client bazat pe datagrame

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

public class DatagramClient {

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

    // Adresa IP si portul la care ruleaza serverul
    InetAddress adresa = InetAddress.getByName("127.0.0.1");
    int port=8200;

    DatagramSocket socket = null;
    DatagramPacket packet = null;
    byte buf[];

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

      // Construim si trimitem pachetul cu cererea catre
         server
      buf = "Duke".getBytes();
      packet = new DatagramPacket(buf, buf.length, adresa,
         port);
      socket.send(packet);

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

      // Afisam raspunsul ("Hello Duke!")
      System.out.println(new String(packet.getData()));
    } finally {
      if (socket != null)
        socket.close();
```

```
    }
  }
}
```

## Trimiterea de mesaje către mai mulţi clienţi




Un grup de clienţi abonaţi pentru trimitere multiplă este specificat prin:

- o adresă IP din intervalul `224.0.0.1` - `239.255.255.255`
- un port UDP

Adresa `224.0.0.0` este rezervată.

Listing 6: Inregistrarea unui client într-un grup

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

public class MulticastClient {

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

    // Adresa IP si portul care reprezinta grupul de clienti
    InetAddress group = InetAddress.getByName("230.0.0.1");
    int port=4444;

    MulticastSocket socket = null;
    byte buf[];

    try {
      // Ne alaturam grupului aflat la adresa si portul
         specificate
      socket = new MulticastSocket(port);
      socket.joinGroup(group);

      // Asteaptam un pachet venit pe adresa grupului
      buf = new byte[256];
      DatagramPacket packet = new DatagramPacket(buf, buf.
         length);

      System.out.println("Asteptam un pachet...");
      socket.receive(packet);

      System.out.println(new String(packet.getData()).trim())
         ;
    } finally {
      if (socket != null) {
        socket.leaveGroup(group);
        socket.close();
      }
    }
  }
}
```

Listing 7: Transmiterea unui mesaj către un grup

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

public class MulticastSend {

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

    InetAddress grup = InetAddress.getByName("230.0.0.1");
    int port = 4444;
    byte[] buf;
    DatagramPacket packet = null;

    // Cream un socket cu un numar oarecare
    DatagramSocket socket = new DatagramSocket(0);
    try {
      // Trimitem un pachet catre toti clientii din grup
      buf = (new String("Salut grup!")).getBytes();
      packet = new DatagramPacket(buf, buf.length, grup, port
          );
      socket.send(packet);

    } finally {
      socket.close();
    }
  }
}
```

## Proxy

Un proxy este un server intermediar aflat între client şi serverul real.
Scopul:

- Imbunătăţirea performanţei
- Securitate (filtrare)

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

## RMI (Remote Method Invocation)

- Programare de reţea la un nivel superior
- Tehnologie Java pentru implementarea aplicaţiilor distribuite
- Oferă o sintaxă şi semantică similare cu cele ale aplicaţiilor ne-distribuite
- Permite colaborarea obiectelor aflate în maşini virtuale diferite.
- Permite unuei aplicaţii să apeleze metode ale unui obiect aflat în alt spaţiu de adrese.
- Portabilitate

## JXTA (Juxtapose)

- Infrastructură pentru comunicare *peer-to-peer*
- Lansată de Sun Microsystems
- Formată dintr-o mulţime de protocoale open-source care permit comunicarea P2P în reţea între dispozitive de orice fel: PC, server, PDA, telefon, etc.
- Peer-urile JXTA formează o reţea virtuală în care pot interacţiona direct, independent de firewall-uri, NAT (network address translation) sau protocoale de transport
- `www.jxta.org`

# JMS (Java Message Service)

*Mesageria* = Formă de comunicare bazată pe mesaje între componente sau aplicaţii (clienţi). Caracteristici:

- **asincronă**
- **loosely-coupled** (spre deosebire de RMI care este *tightly-coupled*)
- clienţii comunică prin intermediul unor *agenţi*

JMS oferă: un API Java pentru crearea trimiterea, primirea, citirea de mesaje în mod *sigur*, *relaxat* şi *asincron*.