Curs 9
Programarea în retea

Notiuni generale despre retele

Programarea în retea implica trimiterea de mesaje si date între aplicatii ce ruleaza pe calculatoare aflate într-o retea locala sau conectate la Internet. Pachetul care ofera suport pentru scrierea aplicatiilor de retea este **java.net**. Clasele din acest pachet ofera o modalitate facila de programare în retea, fara a fi nevoie de cunostine prealabile referitoare la comunicarea efectiva între calculatoare. Cu toate acestea sunt necesare câteva notiuni fundamentale referitoare la retele, cum ar fi protocol, adresa IP, port, socket.

Ce este un protocol ?

Un *protocol* reprezinta o conventie de reprezentare a datelor folosita în comunicarea între doua calculatoare. Având în vedere faptul ca orice informatie care trebuie trimisa prin retea trebuie serializata astfel încât sa poata fi transmisa secvential, octet cu octet, catre destinatie, era nevoie de stabilirea unor conventii (protocoale) care sa fie folosite atât de calculatorul care trimite datele cât si de cel care le primeste.
Cele mai utilizate protocoale sunt **TCP** si **UDP**.

Definitii

TCP (Transport Control Protocol) este un protocol ce furnizeaza un flux sigur de date între doua calculatoare. Acest protocol asigura stabilirea unei conexiuni permanente între cele doua calculatoare pe parcursul comunicatiei.

UDP (User Datagram Protocol) este un protocol ce trimite pachete independente de date, numite *datagrame*, de la un calculator catre altul fara a garanta în vreun fel ajungerea acestora la destinatie. Acest protocol nu stabileste o conexiune permanta între cele doua calculatoare.

Cum este identificat un calculator în retea ?

Orice calculator gazda conectat la Internet este identificat în mod unic de adresa sa **IP** (IP este acronimul de la *Internet Protocol*). Aceasta reprezinta un numar reprezentat pe 32 de biti, uzual sub forma a 4 octeti, cum ar fi de exemplu: `193.226.26.231 si este numit adresa IP numerica. Corespunzatoare unei adrese numerice exista si o adresa IP simbolica, cum ar fi fenrir.infoiasi.ro.`
`De asemenea fiecare calculator aflat într-o retea locala are un nume unic ce poat fi folosit la identificarea locala a acestuia.`

`Clasa Java care reprezinta notiunea de adresa IP este` **`InetAddress.`**

`Ce este un port ?`

`Un calculator are în general o singura legatura fizica la retea. Orice informatie destinata unei anumite masini trebuie deci sa specifice obligatoriu adresa IP a acelei`

masini. Insa pe un calculator pot exista concurent mai multe procese care au stabilite conexiuni în retea, asteptând diverse informatii. Prin urmare datele trimise catre o destinatie trebuie sa specifice pe lânga adresa IP a calculatorului si procesul catre care se îndreapta informatiile respective. Identificarea proceselor se realizeaza prin intermdiul **porturilor**.

Un *port* este un numar de 16 biti care identifica în mod unic procesle care ruleaza pe o anumita masina. Orice aplicatie care realizeaza o conexiune în retea va trebui sa ataseze un numar de port acelei conexiuni. Valorile pe care le poate lua un numar de port sunt cuprinse între 0 si 65535 (deoarece sunt numere reprezentate pe 16 biti), numerele cuprinse între 0 si 1023 fiind însa rezervate unor servicii sistem si, din acest motiv, nu trebuie folosite în aplicatii.

Clase de baza din java.net

Clase din java.net permit comunicare între procese folosind protocoalele TCP si UDP si sunt prezentate în tabelul de mai jos.

TCP	UDP
URL URLConnection Socket ServerSocket	DatagramPacket DatagramSocket MulticastSocket

Aceste clase permit programarea de retea la *nivel de aplicatie*.
Cele 7 nivele ale comunicarii în retea sunt :

Application-level layers	Application Layer
	Presentation Layer
	Session Layer (Sockets)
Data communication-level layers	Transport Layer (TCP, UDP)
	Network Layer (IP)
	Data Layer
	Physical Layer

Lucrul cu URL-uri

Definitie

URL este acronimul pentru *Uniform Resource Locator* si reprezinta o referinta (adresa) la o resursa aflata pe Internet. Aceasta este în general un fisier reprezentând o pagina Web sau o imagine, însa un URL poat referi si interogari la baze de date, rezultate ale unor comenzi (programe), etc.

Exemple de URL-uri sunt:
http://java.sun.com/
http://students.infoiasi.ro/index.html
http://www.infoiasi.ro/~acf/imgs/taz.gif
#url

Dupa cum se observa din exemplele de mai sus, un URL are doua componente principale:

1. Identificatorul protocolului folosit (http, ftp, etc)
2. Numele resursei referite. Acesta are urmatoarele componente:

- numele calculatorului gazda (www.infoiasi.ro)
- calea completa spre resursa referita (~acf/java/curs/9/prog_retea.html)
  Notatia ~user semnifica uzual subdirectorul html al directorului rezervat pe server utilizatorului specificat (HOME).
  In cazul în care este specificat doar un director, fisierul ce reprezinta resursa va fi considerat implicit index.html.
- optional, o referinta de tip *anchor* în cadrul fisierului referit (#url)
- optional, portul la care sa se realizeze conexiunea

Clasa care permite lucrul cu URL-uri este **java.net.URL**. Aceasta are mai multi constructori pentru creearea de obiecte ce reprezinta referinte catre resurse aflate în retea, cel mai uzual fiind cel care primeste ca parametru un sir de caractere. In cazul în care sirul nu reprezinta un URL valid va fi aruncata o exceptie de tipul MalformedURLException.

```
try {
        URL myURL = new URL("http://java.sun.com");
} catch (MalformedURLException e) {
        . . .
}
```

Odata creat, un obiect de tip URL poate fi folosit pentru

- aflarea informatiilor despre resursa referita (numele calculatorului gazda, numele fisierului, protocolul folosit. etc),
- citirea printr-un flux a continutului fisierului respectiv
- conectarea la acel URL pentru citirea si scrierea de informatii

Citirea continutului unui URL

Orice obiect de tip URL poate returna un flux de intrare de tip InputStream pentru citirea continutului sau. Secventa clasica pentru aceasta operatiune este:

```
//Afisarea paginii index.html de la adresa www.infoiasi.ro
public class CitireURL {
  public static void main(String[] args) throws IOException{
        BufferedReader br = null;
        try {
                URL resursa = new URL("http://www.infoiasi.ro");
                InputStream in = resursa.openStream();
                br = new BufferedReader(new InputStreamReader(in));
                String linie;
                while ((linie = br.readLine()) != null) {
                        //proceseaza linia citita
                        System.out.println(linie);
                }
        } catch(MalformedURLException e) {
                System.err.println("URL incorect: " + e);
        }
        finally {
                br.close();
        }
  }
}
```

Conectarea la un URL

Se realizeaza prin metoda **openConnection** ce realizeaza stabilirea unei conexiuni bidirectionale cu resursa specificata. Aceasta conexiune este reprezentata de un obiect de tip **URLConeection** ce permite crearea atât a unui flux de intrare pentru citirea informatiilor de la URL-ul specificat cât si a

unui flux de iesire pentru scrierea de date catre acel URL. Operatiunea de trimitere de date dintr-un program catre un URL este similara cu trimiterea de date dintr-un `FORM` aflat într-o pagina HTML. Metoda folosita pentru trimitere este **POST**. In cazul trimiterii de date, obiectul URL este de fapt un program (comanda) ce ruleaza pe serverul Web referit prin URL-ul respectiv (servlet, cgi-bin, php, etc).

Socket-uri

Definitie
Un *socket (soclu)* este o abstractiune software folosita pentru a reprezenta fiecare din cele doua "capete" ale unei conexiuni între doua procese ce ruleaza într-o retea. Fiecare socket este atasat unui port astfel încât sa poata identifica unic programul caruia îi sunt destinate datele.

Socket-urile sunt de doua tipuri:

- TCP, implementate de clasele `Socket` si `ServerSocket`
- UDP, implementate de clasa `DatagramSocket`

O aplicatie de retea ce foloseste socket-uri se încadreaza în modelul **client/server** de concepere a unei aplicatii. In acest model aplicatia este formata din doua categorii distincte de programe numite *servere*, respectiv *clienti*.
Programele de tip *server* sunt cele care ofera diverse servicii eventualilor clienti, fiind în stare de asteptare atâta vreme cât nici un client nu le solicita serviciile Programele de tip *client* sunt cele care initiaza conversatia cu un server, solicitând un anumit serviciu. Uzual, un server trebuie sa fie capabil sa trateze mai multi clienti simultan si, din acest motiv, fiecare cerere adresata serverului va fi tratata într-un fir de executie separat.

Comunicarea prin conexiuni

In acest model se stabileste o conexiune TCP între un program client si un server care furnizeaza un anumit serviciu.

Structura generala a unui server bazat pe conexiuni

```
               while (true) {
                   accept a connection ;
                   create a thread to deal with the client ;
               end while
import java.net.*;
import java.io.*;

public class SimpleServer extends Thread {

  // Definesc portul pe care se gaseste serverul in afara intervalului 1-1024:
  public static final int PORT = 8100;
  private static ServerSocket serverSocket = null;

  private Socket clientSocket = null;

  public void run() {
         //Executa solicitarea clientului
         String cerere, raspuns;
         try {
                //in este fluxul de intrare de la client
```

```
                BufferedReader in = new BufferedReader(new InputStreamReader(
                        clientSocket.getInputStream() ));

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

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

                //trimit raspuns clientului
                raspuns = "hello " + cerere;
                out.println(raspuns);
                out.flush();

        } catch (IOException e) {
                System.err.println("Eroare de citire/scriere \n" + e);
        } finally {
                // Inchid socketul deschis pentru clientul curent
                try {
                        clientSocket.close();
                } catch (IOException e) {
                        System.err.println("Socketul nu poate fi inchis \n" + e);
                }

        }
  }


  public SimpleServer() throws IOException {

        serverSocket = new ServerSocket(PORT);
        try {
                //Asteapta un client
                clientSocket = serverSocket.accept();

                //Executa solicitarea clientului intr-un fir de executie
                new Thread(this).start();

        } finally {
                serverSocket.close();
        }
  }

  public static void main(String[] args) throws IOException {
        SimpleServer server = new SimpleServer();
  }
}
```

Structura generala a 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 serverAddress = "127.0.0.1";
        //portul la care serverul ofera serviciul
        int PORT = 8100;

        Socket clientSocket = null;
```

```
            PrintWriter out = null;
            BufferedReader in = null;
            String cerere, raspuns;

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

                    //se trimite o cerere la server
                    cerere = "duke";
                    out.println(cerere);

                    //se asteapta raspuns de la server
                    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 (clientSocket!= null)
                            clientSocket.close();
            }
    }
}
```

Comunicarea prin datagrame

In acest model clientul trimite un pachet cu cererea catre server, acesta primeste pachetul si returneaza raspunsul tot prin intermediul unui pachet. Un astfel de pachet se numeste *datagrama* si este reprezentat printr-un obiect din clasa **DatagramPacket**. Primirea si trimiterea datagramelor se realizeaza tot prin intermediul unui socket, acesta fiind modelat preintr-un obiect al clasei **DatagramSocket**.

Structura generala a 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 DatagramServer() throws IOException {

          Socket = new DatagramSocket(PORT);
          try
          {
                  while (true) {
```

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

                        //Astepta aparitia unui pachet cu cererea
                        socket.receive(cerere);

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

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

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

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

Structura generala a 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 address = InetAddress.getByName("127.0.0.1");
        int port=8200;

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

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

                //Construieste si trimite pachetul cu cerere catre server
                buf = "Duke".getBytes();
                packet = new DatagramPacket(buf, buf.length, address, port);
                socket.send(packet);

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

                //Afiseaza raspunsul
```

```
                System.out.println(new String(packet.getData()));

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

Trimiterea de mesaje catre mai multi clienti

Diverse situatii impun gruparea mai multor clienti astfel încât un mesaj (pachet) trimis pe adresa grupului sa fie receptionat de fiecare dintre acestia. Gruparea mai multor programe în vederea trimiterii multiple de mesaje se realizeaza prin intermediul unui socket special, descris de clasa **MulticastSocket**, extensie a clasei `DatagramSocket`.

Un grup de clienti abonati pentru trimitere multipla este specificat printr-o adresa IP din intervalul `224.0.0.1 - 239.255.255.255` si un port UDP. Adresa `224.0.0.0` este rezervata si nu trebuie folosita.

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

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

                System.out.println(new String(packet.getData()));

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

Transmiterea unui mesaj catre un grup

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

```
import java.io.*;

public class MulticastSend {

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

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

          //Creeaza un socket cu un numar oarecare
          DatagramSocket socket = new DatagramSocket(0);

          try
          {
                 //Trimite un pachet catre toti clientii din grup
                 buf = (new String("Salut grup")).getBytes();
                 packet = new DatagramPacket(buf, buf.length, group, port);
                 socket.send(packet);

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