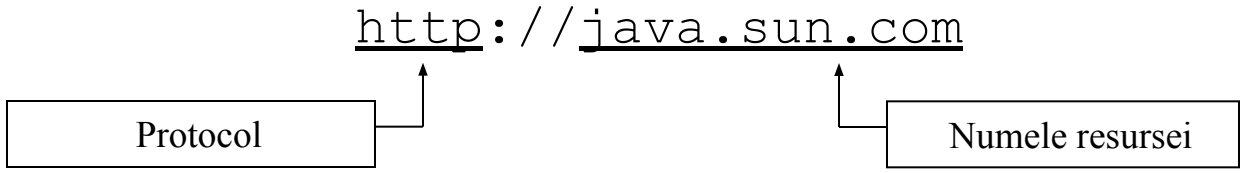
Programare in retea

Programare in retea

1.1 Folosirea URL-urilor

- URL (Uniform Resource Locator) reprezinta o referinta (adresa) a unei resurse de pe Internet
- Structura unui URL: *hostname*, numele fisierului, numarul portului (optional) si numele resursei (optional)



- Accesarea unui URL se poate face prin intermediul unui obiect Java din pachetul *java.net*

Programare in retea

1.1 Folosirea URL-urilor

```
   import java.net.*;
   import java.io.*;
   public class ParseURL {
      public static void main(String[] args) throws Exception {
      URL aURL = new URL("http://java.sun.com:80/docs/books/" +
   "tutorial/index.html#DOWNLOADING");
System.out.println("protocol="+ aURL.getProtocol());
   System.out.println("host = " + aURL.getHost());
   System.out.println("filename = " + aURL.getFile());
   System.out.println("port = " + aURL.getPort());
   System.out.println("ref = " + aURL.getRef());
      }
   }
```

Rezultat:

protocol = http
host = java.sun.com
filename = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING

Programare in retea

1.2 Connectarea la o resursa web

- Accesarea unei resurse se face prin intermediul obiectului URL
- Metoda *openStream* intoarce un obiect *java.io.InputStream* care se poate accesa ca orice alt flux de intrare

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

public class ReadURL {
   public static void main(String[] args) throws Exception {
       URL osu = new URL("http://www.upm.ro/");
    BufferedReader in = new BufferedReader(
                 new InputStreamReader(osu.openStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
        System.out.println(inputLine);
    in.close();
    }
}
```

Programare in retea

1.2 Connectarea la o resursa web

- O alta metoda a obiectului URL este *openConnection* care intoarce un obiect de tip *URLConnection* – o conexiune catre resursa accesata
- *URLConnection* permite atat citirea cat si scrierea in cadrul resursei

```
try {
    URL osu = new URL("http://www.upm.ro/");
    URLConnection osuConnection = osu.openConnection();
} catch (MalformedURLException e) { // new URL() failed
    . . .
} catch (IOException e) {
    . . .
}
```

Programare in retea

1.4 Programarea serverului

- Pasi necesari:
  1. Se creaza un obiect de tip *ServerSocket*:
     ```
     ServerSocket server=new ServerSocket(port,queueLength);
     ```
  2. Serverul va incepe sa asculte mesaje:
     ```
     Socket connection = server.accept();
     ```
  3. Se initializeaza obicte de tip *OutputStream* si *InputStream* care permit serverului sa comunice cu clientul:
     ```
     InputStream input = connection.getInputStream();
     OutputStream output = connection.getOutputStream();
     ```
  4. Faza de procesare: clientul si serverul comunica prin intermediul obiectelor de tip *InputStream* si *OutputStream*
  5. Dupa ce comunicarea se incheie, serverul inchide conexiunea prin apelarea metodei *close()* la nivelul socketului si pentru fluxurile corespunzatoare

Programare in retea

1.3 Socket-uri

- Pentru unele aplicatii este nevoie de stabilirea unui canal de comunicatie *low-level*, ca in cazul unei aplicatii de tip client-server.
- Un *socket (soclu)* este o abstractiune software folosita pentru a reprezenta fiecare din cele doua "capete" ale unei conexiuni intre doua procese ce ruleaza intr-o retea. Fiecare socket este atasat unui port astfel incat sa poata identifica unic programul caruia ii sunt destinate datele.
- Pentru a se conecta, atat clientul cat si serverul au nevoie de cate un socket pentru scriere, respectiv pentru citire.
- Socket-urile sunt de doua tipuri:
  - TCP (Transport Control Protocol), implementate de clasele *Socket* si *ServerSocket*
  - UDP (User Datagram Protocol), implementate de clasa *DatagramSocket*
- Aceste clase se gasesc in pachetul *java.net.*

Programare in retea

1.5 Programarea clientului

- Pasi necesari:
  1. Se creaza un obiect de tip *Socket*:
     ```
     Socket connection = new Socket (serverAddress, port);
     ```
  2. Se initializeaza obicte de tip *OutputStream* si *InputStream* care permit serverului sa comunice cu serverul
  3. Faza de procesare: clientul si serverul comunica prin intermediul obiectelor de tip *InputStream* si *OutputStream*
  4. Dupa ce comunicarea se incheie, clientul inchide conexiunea

Programare in retea

1.6 Aplicatie client-server

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

class Server {
   public static void main(String args[]) {
      String data = "Mesaj!";
      try {
         ServerSocket server_socket = new ServerSocket(1234);
         System.out.println("Pornesc...");

         Socket socket = server_socket.accept();

         System.out.print("Sunt functional!\n");
         PrintWriter outToClient = new
PrintWriter(socket.getOutputStream(), true);

         System.out.print("Trimit mesaj: " + data + "\n");
         outToClient.print(data);

         outToClient.close();
         socket.close();
         server_socket.close();
      }
      catch(Exception e) {
         System.out.print("Oops! Nu a functionat!\n");
      }
   }
}
```

Programare in retea

1.6 Aplicatie client-server

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

class Client {
   public static void main(String args[]) {
       try {
         Socket socket = new Socket("localhost", 1234);

         BufferedReader inFromServer = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));

        System.out.print("Primesc mesaj: ");
        while (!inFromServer.ready()) {}

         System.out.println(inFromServer.readLine());

         inFromServer.close();
        }
        catch(Exception e) {
         System.out.print(" Oops! Nu a functionat!\n");
      }
   }
}
```

Programare in retea

1.7 Datagrame

- Protocolul UDP furnizeaza o alta metoda de comunicare in retea, prin intermediul *datagramelor*.
- In acest model clientul trimite un pachet cu cererea catre server, acesta primeste pachetul si returneaza raspunsul tot prin intermediul unui pachet.
- Clasa *DatagramSockets* se foloseste pentru realizarea conexiunii

Programare in retea

1.7 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();
```

Programare in retea

1.7 Datagrame

```
            //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();
        }
    }
```

Programare in retea

1.7 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);
```

Programare in retea

1.7 Datagrame

```
            //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();
        }
    }
}
```