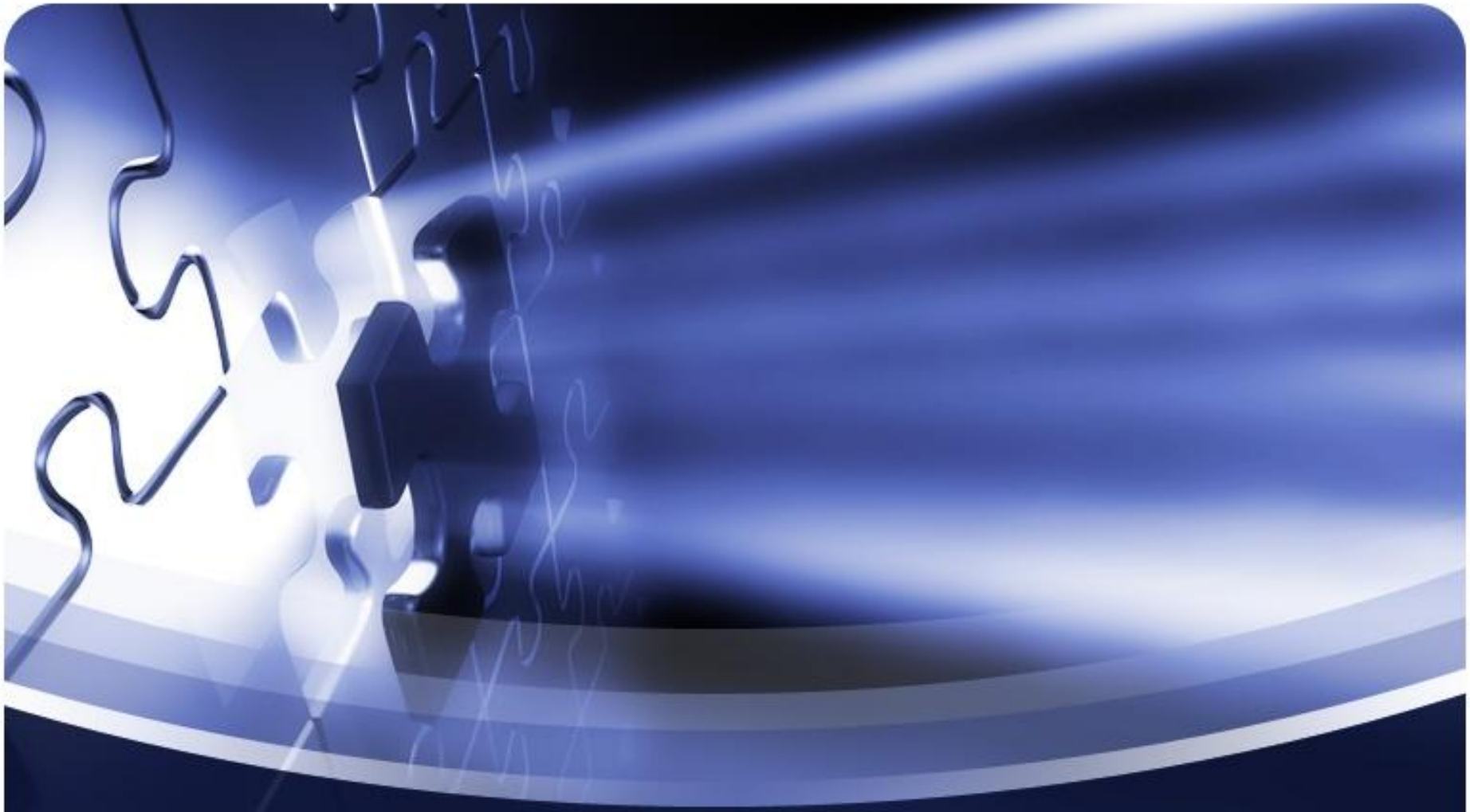


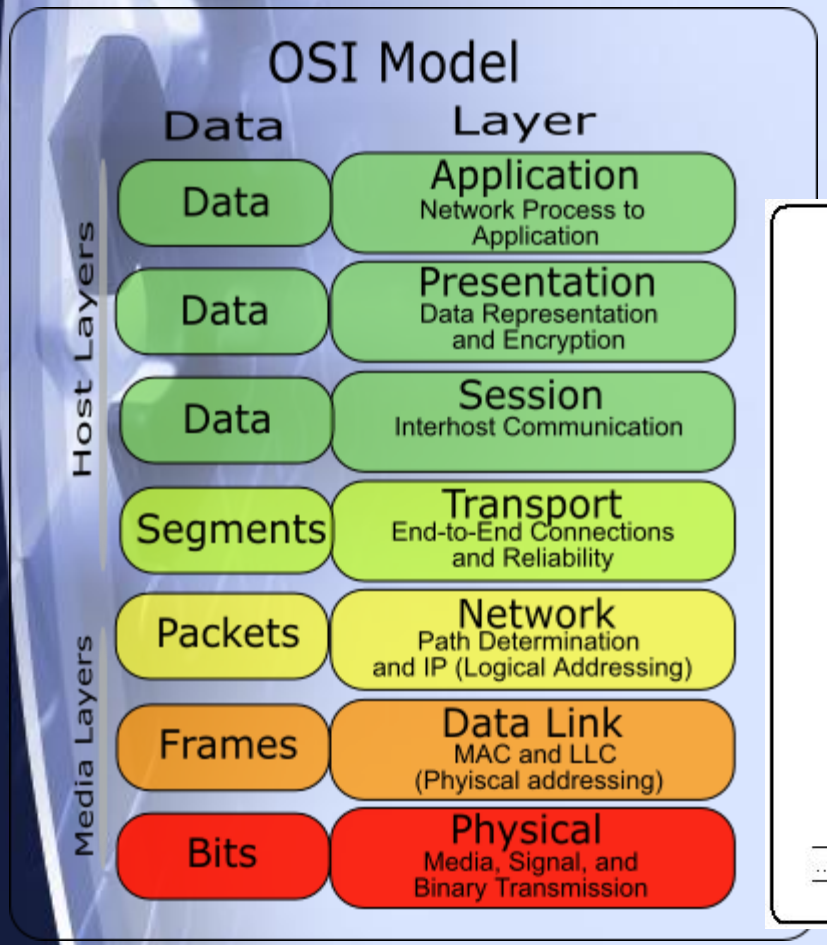
Mrežno programiranje

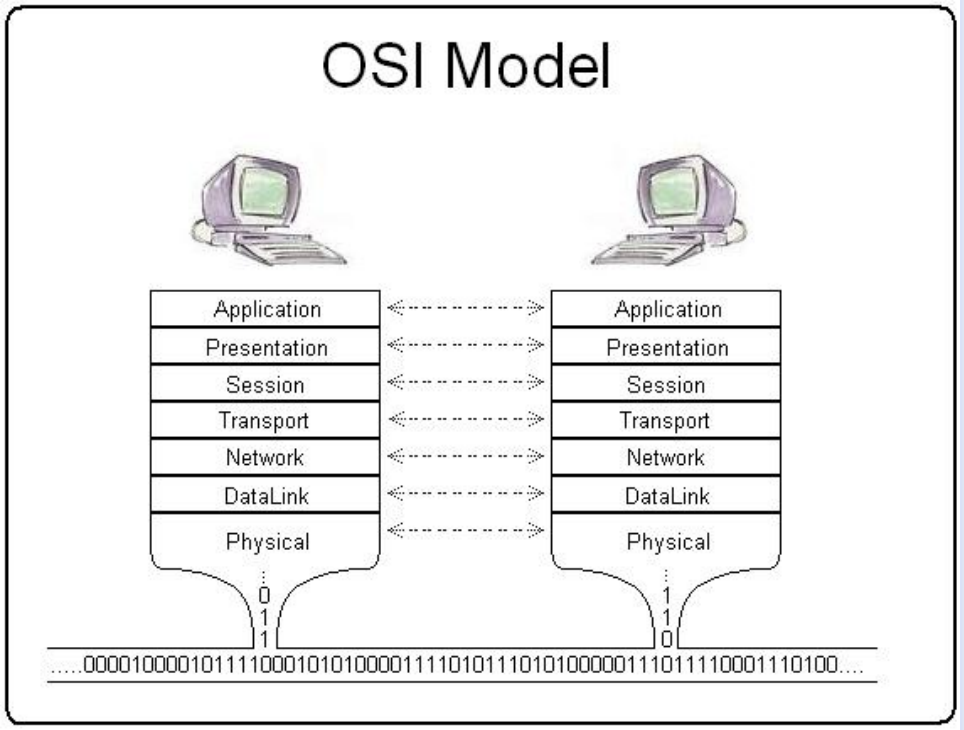
Programski jezici II

Osnovne odrednice

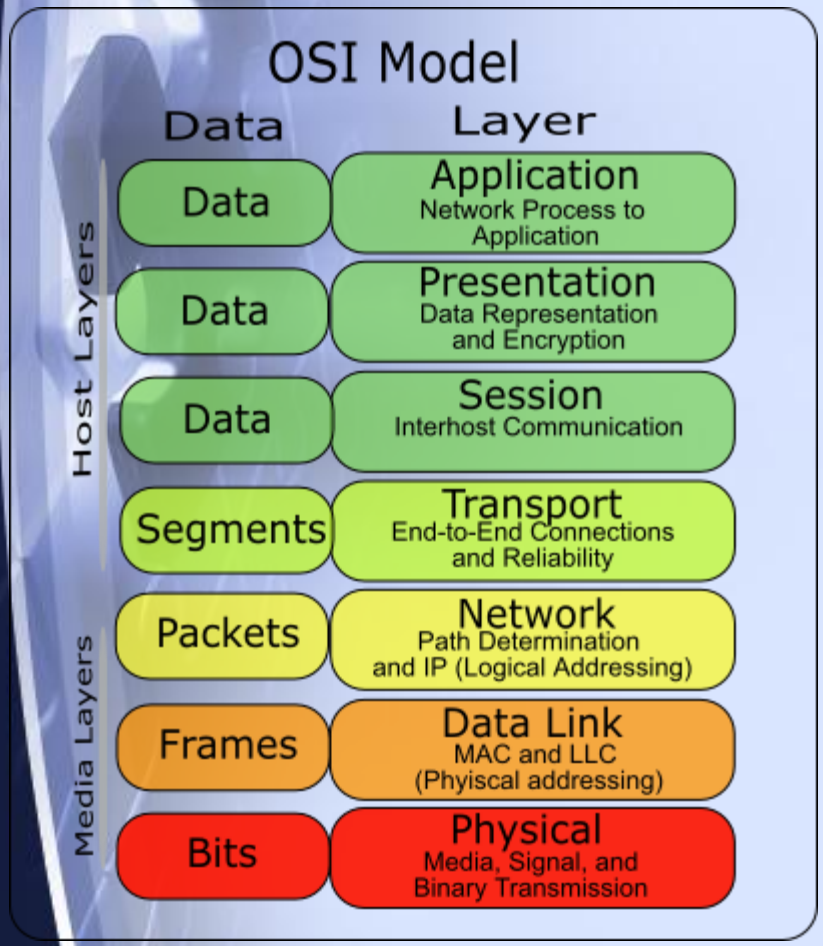
- mrežno programiranje – pisanje programa koji komuniciraju preko računarske mreže
- oslanja se na IP mrežni protokol
- mogućnost korišćenja TCP i UDP protokola
- komunikacija se odvija kroz stream-ove (stream za čitanje i stream za pisanje); jednako kao i sa fajlovima u okviru fajl-sistema; dvosmjerna komunikacija
- paket: java.net
  - API niskog nivoa
  - API visokog nivoa

OSI referentni model





OSI referentni model



- **HTTP, SMTP, FTP, POP3, IMAP**
- **SSL/TLS**
- **TCP, UDP**
- **IP**
- **Ethernet**

IP adresa

- identifikator čvora u IP mreži je IP adresa
- IPv4 – 32-bitni broj – 4 grupe po 8 bita
- radi lakšeg pamćenja – 4 decimalno zapisana okteta, npr. 147.91.197.2
- pomoću simboličke adrese, npr. www.etfbl.net
- klasa InetAddress predstavlja adresu čvora u mreži

```
InetAddress a = InetAddress.getByName("www.etfbl.net");
InetAddress b = InetAddress.getByName("147.91.197.2");
InetAddress c = InetAddress.getLocalHost();
```

- IPv6 – 128-bitni
  - Otklanja nedostatke IPv4:
    - Ograničen adresni prostor
    - Sigurnost

InetAddress

```
class InetAddressTest {
   public static void main(String args[]) throws IOException
   {
      System.out.print("localhost IP adresa: ");
      InetAddress addr = InetAddress.getLocalHost();     // 1
      System.out.println(addr.getHostAddress());         // 2
      System.out.println("======================");
      System.out.print("etfbl.net IP adresa: ");
      addr = InetAddress.getByName("etfbl.net");         // 3
      System.out.println(addr.getHostAddress());         // 4
      System.out.print("etfbl.net FQDN: ");
      System.out.println(addr.getCanonicalHostName());   // 5
      System.out.print("etfbl.net je dostupan: ");
      System.out.println(addr.isReachable(5000));        // 6
      System.out.println("======================");
      System.out.println("www.google.com IP adrese:");
      InetAddress addrs[] =
InetAddress.getAllByName("www.google.com");              // 7
      for (int i = 0; i < addrs.length; i++)
         System.out.println(addrs[i].getHostAddress()); // 8
   }
}
```

InetAddress

```
localhost IP adresa: 192.168.1.100
======================
etfbl.net IP adresa: 147.91.197.2
etfbl.net FQDN: nucleus.etfbl.net
etfbl.net je dostupan: true
======================
www.google.com IP adrese:
74.125.39.103
74.125.39.104
74.125.39.105
74.125.39.106
74.125.39.147
74.125.39.99
```

Port

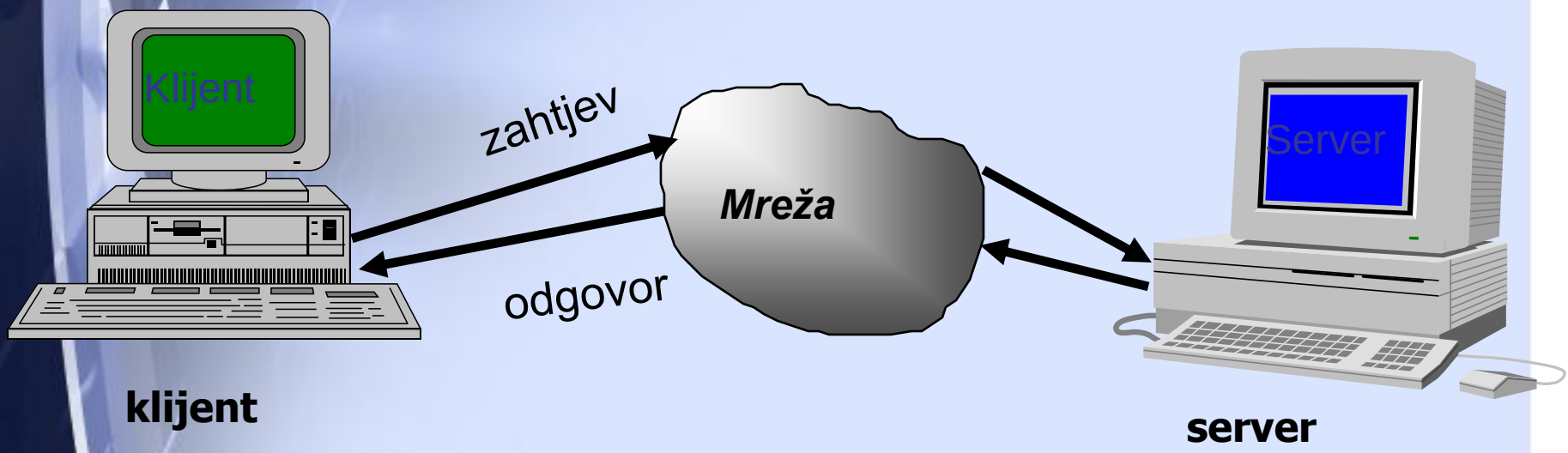
- 16-bita – 65536 – 0 do 65535
- 0 – 1023 – well known ports

- http://www.iana.org/assignments/port-numbers

- FTP 21
- DNS 53
- SMTP 25
- HTTP 80
- POP3 110
- SNMP 161
- HTTPS 443

Elementi klijent-server arhitekture

- klijent, server, mreža



Klase za mrežno programiranje

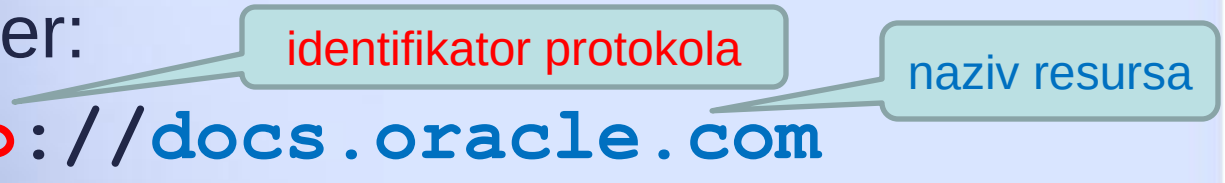
- TCP i UDP protokoli
- TCP:
  - URL
  - URLConnection
  - Socket
  - ServerSocket
- UDP
  - DatagramPacket
  - DatagramSocket
  - MulticastSocket

URL

- URL – Uniform Resource Locator
- predstavlja referencu (adresu) na resurs koji se nalazi na Internetu (ili drugoj mreži)
- web čitači
- Java programi mogu koristiti URL kako bi pronašli i pristupili resursima na Internetu
- klasa URL u paketu java.net
  - služi za reprezentaciju URL adrese
- dvojako značenje:
  - URL adresa
  - URL objekat

URL

- primjer:



- format naziva resursa zavisi od protokola
- najčešće se sadrži sljedeće komponente:
  - Host Name – naziv računara na kojem se resurs nalazi
  - Port Number – port na kojem server osluškuje, obično je opcion
  - Filename – putanja do datoteke

Kreiranje URL objekta

- najjednostavniji način – na bazi stringa koji predstalja URL adresu
- primjer: http://www.etfbl.net

```
URL etf = new URL(“http://www.etfbl.net”);
```

- ovaj URL predstavlja apsolutnu putanju – sadrži sve informacije neophodne za pristup specificiranom resursu
- URL objekti se mogu kreirati i na osnovu relativnih URL adresa

```
URL etf = new URL(“http://www.etfbl.net”);
URL etfJava = new URL(etf, “java/index.html”);
URL etfIP = new URL(etf, “IP/index.html”);
```

URL konstruktori

- URL(String spec)
  - kreira URL objekat na bazi datog Stringa
- URL(String protocol, String host, int port, String file)
  - kreira URL objekat na bazi datog protokola, naziva hosta, porta i putanje do resursa
- URL(String protocol, String host, int port, String file, URLStreamHandler handler)
  - kreira URL objekat na bazi datog protokola, naziva hosta, porta, putanje do resursa i URLStreamHandler-a
- URL(String protocol, String host, String file)
  - kreira URL objekat na bazi datog protokola, naziva hosta i putanje do resursa
- URL(URL context, String spec)
  - kreira URL objekat parsirajući dati string unutar datog konteksta
- URL(URL context, String spec, URLStreamHandler handler)
  - kreira URL objekat parsirajući dati string sa URLStreamHandler-om, unutar datog konteksta

- svi konstruktori mogu baciti izuzetak MalformedURLException
  - potrebno je uhvatiti ga u try/catch bloku

Parsiranje URL-a

- metode:
  - getProtocol
  - getAuthority
  - getHost
  - getPort
  - getPath
  - getQuery
  - getFile
  - getRef
- primjer:
  - protocol = http
  - authority = www.etfbl.net:80
  - host = www.etfbl.net
  - port = 80
  - path = /java/
  - query = c=prikazi&objekat=sadrzaj
  - filename = /java/?c=prikazi&objekat=sadrzaj
  - ref = DOWNLOAD

Čitanje iz URL-a

- nakon kreiranja URL objekta – poziv metode openStream() – vraća stream iz kojeg se može čitati sadržaj sa specificiranog URL-a
- openStream() metoda vraća java.io.InputStreamObject

```
URL etf = new URL("http://www.etfbl.net/");
BufferedReader in = new BufferedReader( new
InputStreamReader( etf.openStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
```

URL

```
public class URLTest {
   public static void main(String[] args) throws IOException
   {
      URL url = new URL("http://www.etfbl.net:80/java/
?objekat=sadrzaj#DOWNLOAD");                                  // 1
      System.out.println("protocol: " + url.getProtocol());
      System.out.println("autority: " + url.getAuthority());
      System.out.println("host: " + url.getHost());
      System.out.println("port: " + url.getPort());
      System.out.println("path: " + url.getPath());
      System.out.println("query: " + url.getQuery());
      System.out.println("filename: " + url.getFile());
      System.out.println("ref: " + url.getRef());

      URL url2 = new URL("http://www.etfbl.net");             // 2
      BufferedReader in = new BufferedReader( new
InputStreamReader( url2.openStream()));                       // 3
      String inputLine;
      while ((inputLine = in.readLine()) != null)             // 4
            System.out.println(inputLine);
      in.close();
   }
}
```

URL

```
protocol = http
authority = www.etfbl.net:80
host = www.etfbl.net
port = 80
path = /java/
query = objekat=sadrzaj
filename = /java/?objekat=sadrzaj
ref = DOWNLOAD
<HTML>
<TITLE>Elektrotehnicki fakultet - Banjaluka</TITLE>
...
</HTML>
```

URLConnection

- apstraktna klasa
- roditeljska klasa svih klasa koje predstavljaju komunikacioni link između aplikacije i servisa koji se izvršava na URL adresi predstavljenoj objektom klase URL
- instance klasa koje nasljeđuju URLConnection klasu mogu se koristiti:
  - za upis u resurs,
  - za čitanje iz resursa specificiranog objektom klase URL, kao i
  - za pristup osobinama udaljenog resursa

Povezivanje na URL - URLConnection

- nakon kreiranja URL objekta – pozivom openConnection metode dobija se URLConnection objekat ili objekat jedne od protokol-specifičnih podklasa, npr. java.net.HttpURLConnection
- objekat tipa URLConnection se koristi za podešavanje parametara i opštih osobina zahtjeva koje je u nekim slučajevim potrebno podesiti prije konekcije
- konekcija se uspostavlja prilikom poziva URLConnection.connect metode
- nije neophodno eksplicitno pozivati connect metodu u svim slučajevima:
  - metode koje zahtjevaju postojanje konekcije, poput getInputStream, getOutputStream će implicitno uspostaviti konekciju

```
try {
   URL etf = new URL("http://www.etfbl.net/");
   URLConnection etfConnection = etf.openConnection();
   etfConnection.connect();
} catch (MalformedURLException e) {
   // new URL() nije uspješan. . .
} catch (IOException e) {
   // openConnection() nije uspješna. . .
}
```

Čitanje iz i upis u URL konekciju

- input stream se dobija iz URLConnection objekat pozivom metode getInputStream
- isti efekat kao i direktno čitanje iz URL-a, ali može biti korisnije jer se URLConnection objekat može koristiti i za druge poslove
- primjer:
  - URLConnectionReader

- upis u URL konekciju

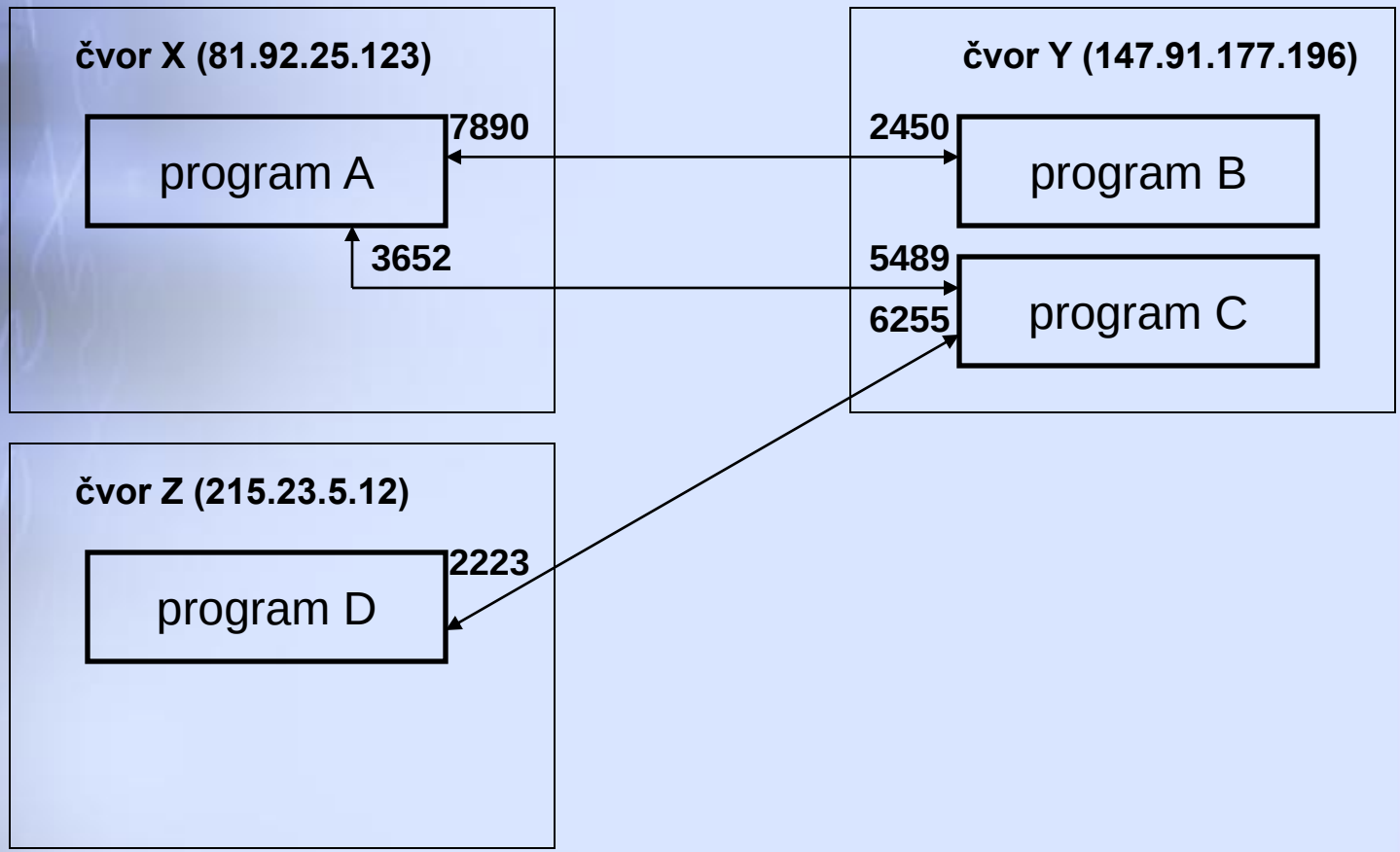
```
URLConnection connection = url.openConnection();
connection.setDoOutput(true);
OutputStreamWriter out = new OutputStreamWriter(
connection.getOutputStream());
out.write("string=" + xyz);
out.close();
```

Pojam Socket-a

- pojam socket-a: uređeni par (IP adresa, port) jednog učesnika u komunikaciji
- uspostavljena veza između dva programa – skup od dva socket-a

Pojam Socket-a

- veza između 2 programa, ne 2 računara



Klasa Socket

- objekti Socket klase predstavljaju uspostavljene TCP konekcije
- otvaranje konekcije:

```
Socket s = new Socket(addr, 80);//addr-InetAddress
Socket s = new Socket("www.etfbl.net", 80);
```

- metoda getInputStream()
- metoda getOutputStream()

Klasa ServerSocket

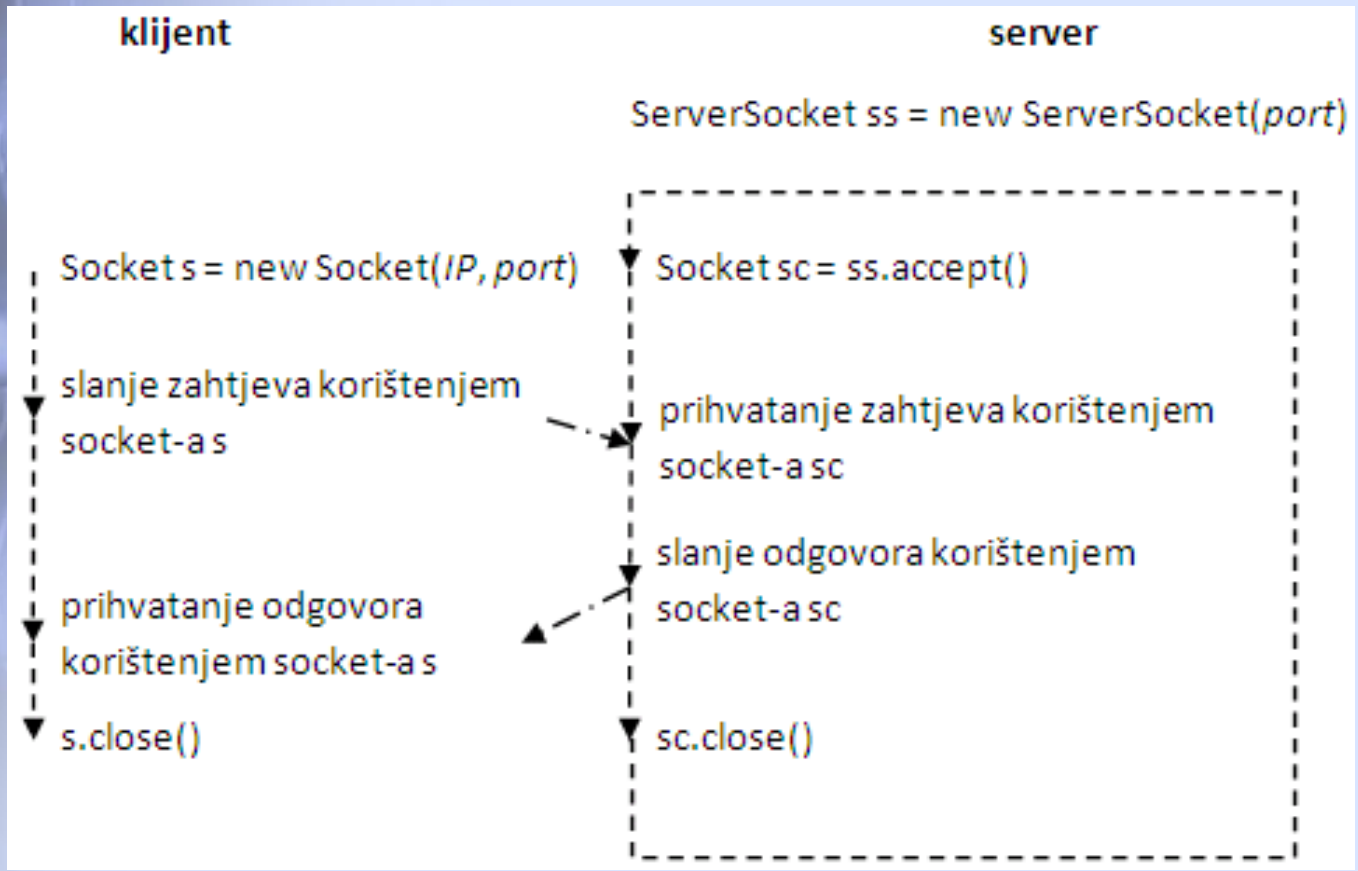
- predstavlja serverski socket
- metoda accept() blokira izvršenje sve dok neki klijent ne uspostavi vezu; tada vraća konstruisan Socket objekat preko koga se komunicira sa klijentom
- kreiranje objekata klase ServerSocket:

```
ServerSocket ss = new ServerSocket(2345);
```

- osluškivanje:

```
ss.accept();
```

Klijent-server komunikacija



Tipičan tok komunikacije – server strana

```
// čekam klijenta...
ServerSocket ss = new ServerSocket(port);
Socket s = ss.accept();

// inicijalizacija
BufferedReader in = new BufferedReader(...,s);
PrintWriter out = new PrintWriter(...,s);

// komunikacija
String request = in.readLine();   // čitam zahtjev
out.println("odgovor");           // šaljem odgovor

// prekid veze
in.close();
out.close();
s.close();
```

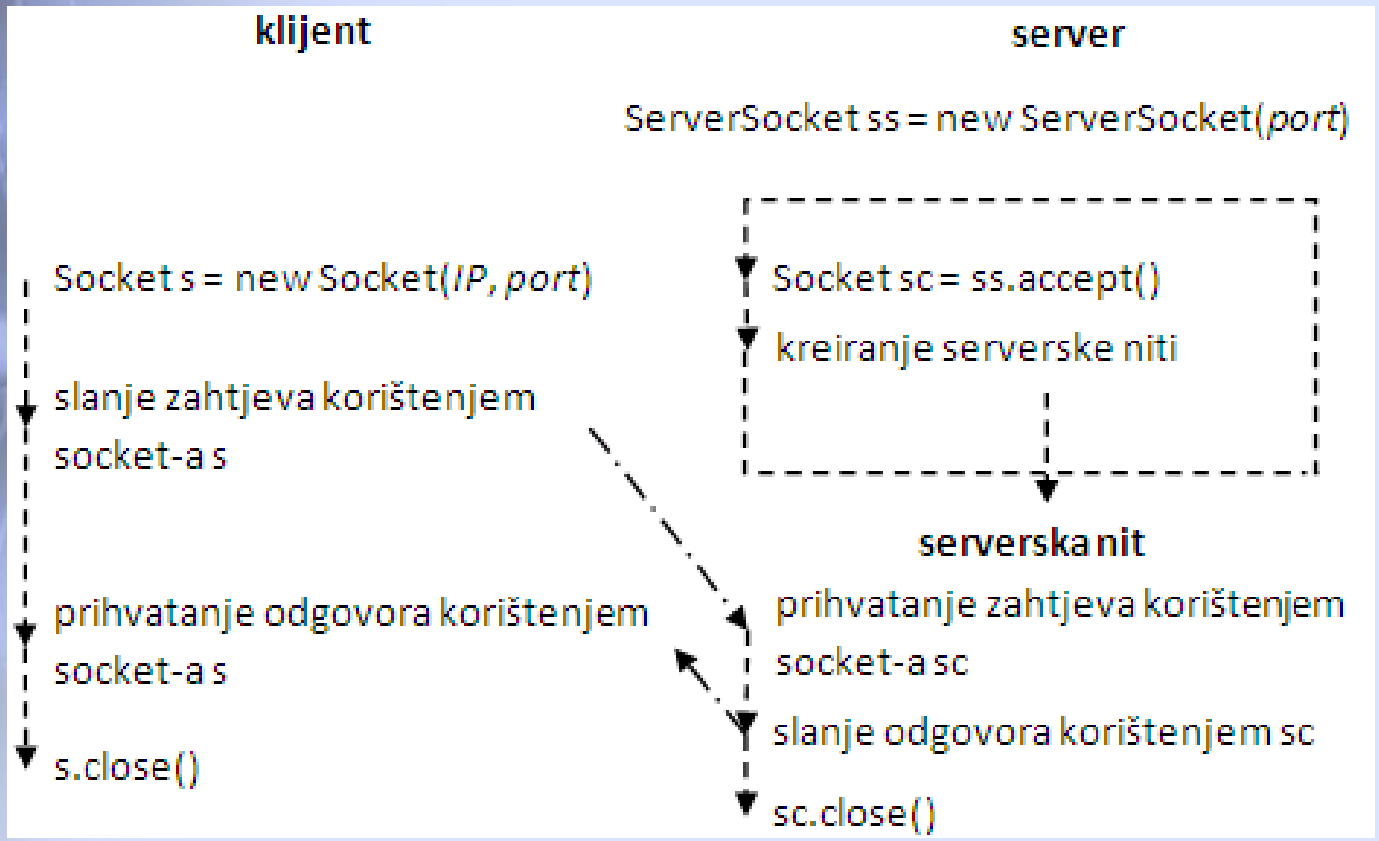
Tipičan tok komunikacije – klijent strana

```
// inicijalizacija
Socket s = new Socket(addr, port);
BufferedReader in = new BufferedReader(...,s);
PrintWriter out = new PrintWriter(...,s);

// komunikacija
out.println("zahtjev"); // šaljem zahtjev
String response = in.readLine();// čitam odgovor

// prekid veze
in.close();
out.close();
s.close();
```

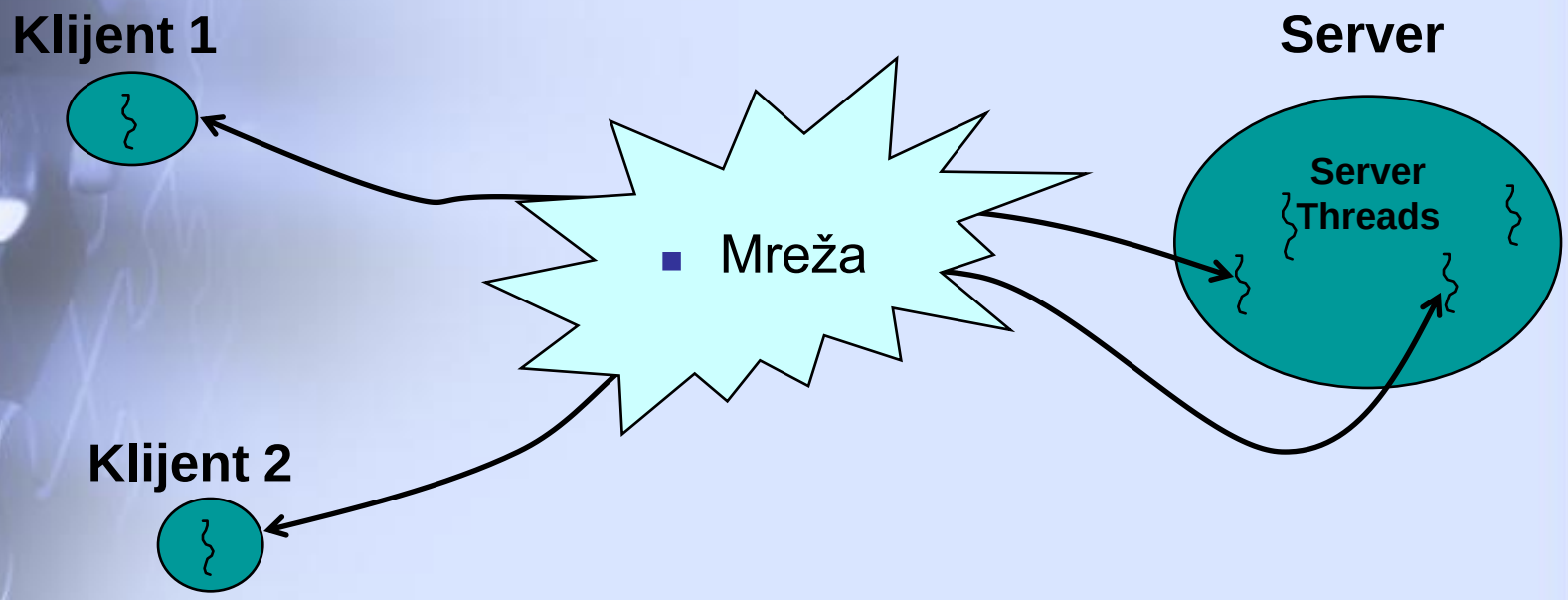
Server koji opslužuje više klijenata



Server koji opslužuje više klijenata

```
// beskonačna petlja
while (true)
   server “sluša”
   prihvata konekciju
   kreira nit za komunikaciju sa klijentom
end while
```

Server koji opslužuje više klijenata

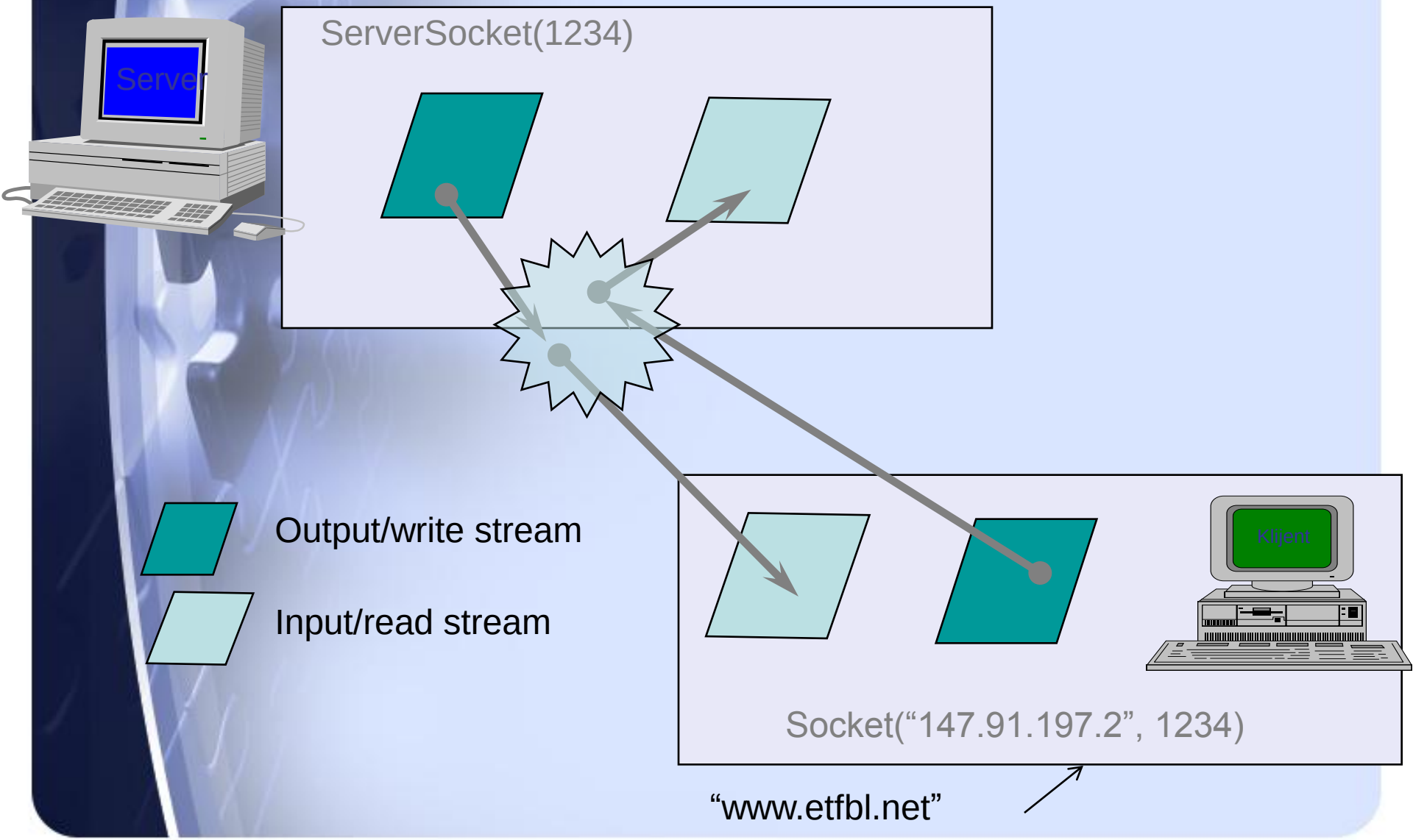


Server koji opslužuje više klijenata

```
// Serverska petlja
ServerSocket ss = new ServerSocket(port);
while (true) {
  Socket s = ss.accept();
  ServerThread st = new ServerThread(s);
}

// poseban thread - obrada pojedinačnog zahtjeva
class ServerThread extends Thread {
  public void run() {
    // inicijalizacija
    // komunikacija
    // prekid veze
  }
}
```

Java Socket-i



Primjer – ECHO server

- klijent
  - klijent uspostavlja vezu sa serverom
  - šalje zahtjev (tekst)
  - čita odgovor servera
  - ispisuje odgovor na konzolu
  - završava komunikaciju
- server
  - čeka klijente u beskonačnoj petlji
  - za svakog klijenta pokreće poseban thread za obradu:
    - čita zahtjev klijenta
    - šalje odgovor – identičan zahtjevu

Datagrami

- datagram je nezavisna, samosadržavajuća poruka poslata preko mreže, čije se stizanje, vrijeme stizanja i sadržaj ne garantuje
- 3 klase za rad sa datagramima:
  - DatagramSocket
  - DatagramPacket
  - MulticastSocket

Datagrami

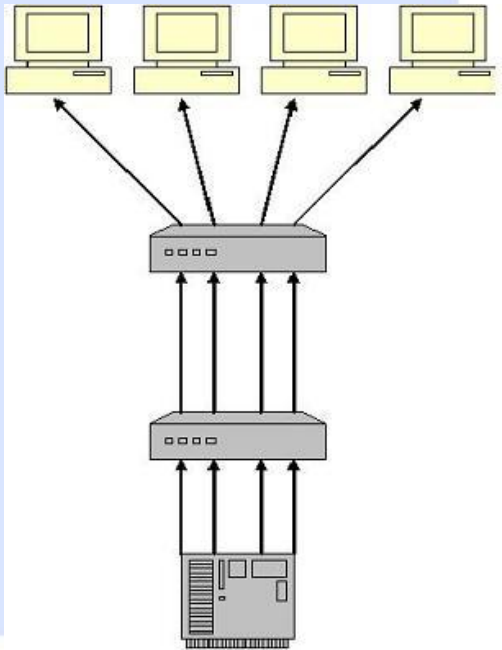
- kreiranje Datagram socket-a
  - na serverskoj strani
    - DatagramSocket socket = new DatagramSocket(4445);
  - na klijentskoj strani
    - DatagramSocket socket = new DatagramSocket();
- kreiranje Datagram paketa
  - za čitanje
    - byte[] buf = new byte[256];
    - DatagramPacket packet = new DatagramPacket(buf, buf.length);
  - za slanje
    - byte[] buf = new byte[256];
    - DatagramPacket packet = new DatagramPacket(buf, buf.length, address, port);
- prihvatanje paketa
  - socket.receive(packet);
- slanje paketa
  - socket.send(packet);
- primjer
  - QuoteServer

Multicast

- slanje ka većem broju klijenata
- http://www.iana.org/assignments/multicast-addresses/multicast-addresses.txt
- Primjer:
  - MulticastServer
  - MulticastClient

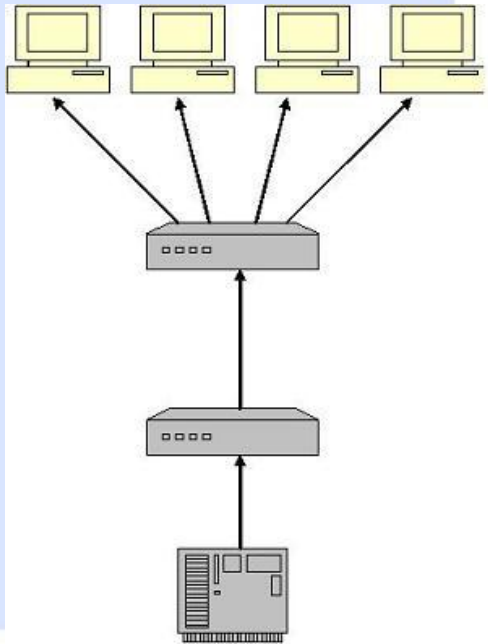
Multicast

- *one-to-many* komunikacija
  - više unicast sesija
  - količina saobraćaja = broj korisnika x broj poruka (u bajtima)
  - na slici: 1 poruka ka 4 klijenta
    - 4 kopije iste poruke



Multicast

- *one-to-many* komunikacija
  - *multicast*
  - prenos podataka ka višestrukim odredištima, simultano, koristeći najefikasniju strategiju prenosa
    - poruke se kopiraju kada se link ka destinacijama dijeli



  - na slici: 1 poruka ka 4 klijenta
    - 1 kopija iste poruke

Multicast

- Kada host šalje podatak prema multicast grupi, potrebno je da podatke stavlja u multicast datagrame, tj. UDP datagrame koji su adresirani prema multicast grupi
- Život UDP datagrama određen je pomoću Time-To-Live vrijednosti (0 - 255)
- 224.0.0.0 – 239.255.255.255

Multicast

- DatagramSocket može se koristiti za:
  - slanje i prijem unicast datagrama
  - slanje i prijem broadcast datagrama
  - slanje multicast datagrama
- MulticastSocket se koristi za:
  - prijem multicast datagrama

Multicast i serijalizacija

- Serijalizacija objekata – bazirana na ObjectOutput**Stream** i ObjectInput**Stream** klasama
- ObjectOutputStream upisuje stream zaglavlje (koje sadrži magic number i version number) u stream pri njegovoj konstrukciji
- ObjectInputStream čita i provjerava ove podatke pri konstrukciji
- zato se i ObjectInputStream konstruktor blokira sve dok ObjectOutputStream ne bude konstruisan na strani pošiljaoca (ObjectOutputStream se uvijek vezuje za socket na odlaznoj strani, prije nego što se ObjectInputStream vezuje za socket na prijemnoj strani)
- Da bi se multicast-om slali objekti stream header informacije moraju biti u svakom datagramu
  - najjednostavnije: kreiranje novog ObjectOutputStream objekta za svaki datagram koji šaljemo, i kreiranje novog ObjectInputStream objekta za svaki datagram koji primamo