OBJEKTNO PROGRAMIRANJE 2

Oznaka predmeta: OP2

Predavanje broj: 09

Nastavna jedinica: JAVA

Nastavne teme:

JOptionPane. Horizontalni i vertikalni klizači. JScrollPane. JTabbedPane. Mrežno programiranje. Socket. Identifikacija čvorova mreže. Komunikacija na klijentskoj i serverskoj strani. Korišćenje niti na serverskoj strani komunikacije.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Eckel B., *Thinking in Java*, 2nd edition, Prentice-Hall, New Jersey 2000.

Cay S. Horstmann and Gary Cornell*: "Core Java, Advanced Features", Vol. 2, Prantice Hall, 2013.*

The Java Tutorial, Sun Microsystems 2001. *http://java.sun.com*

Branko Milosavljević, Vidaković M, *Java i Internet programiranje*, GInT, Novi Sad 2002.

JOptionPane

- Klasa JOptionPane sadrži statičke metode kojima realizuje jednostavne dijaloge u swing-u. Najčešće se koristi kao:
  - dijalog za unos teksta

```
String str = JOptionPane.showInputDialog(null,
                                         "Unesite ime!");
```

  - dijalog prikaza poruke

```
str+=" dobrodosli !";
JOptionPane.showMessageDialog(null,str);
```

  - dijalog izbora

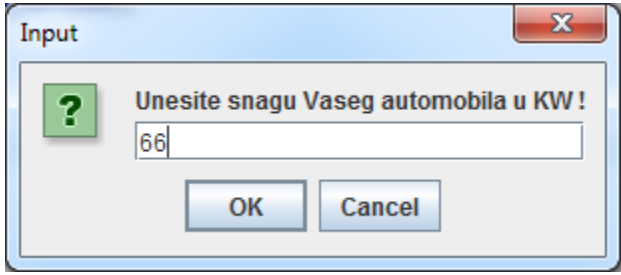
```
int odgovor = JOptionPane.showConfirmDialog(null,
                          "odustajete li ?",
                          "NAZIV DIJALOGA",
                          JOptionPane.YES_NO_OPTION);
```

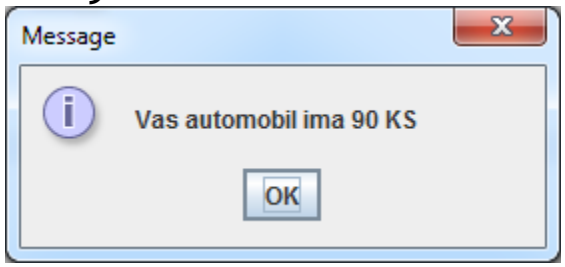
- Sledi primer korišćenja klase JOptionPane.

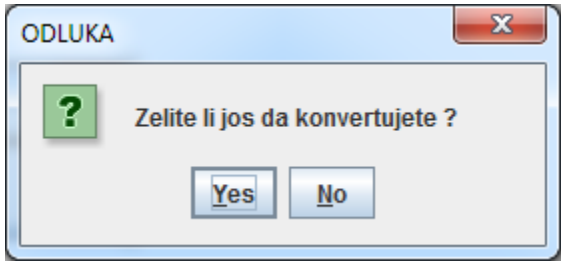
```
import javax.swing.*;
public class Jop {
  public static void main (String[]args){
    String strsnaga="";
    double snaga=0;
```

JOptionPane

```
    int odgovor;
    do{
      strsnaga = JOptionPane.showInputDialog(null,
                 "Unesite snagu Vaseg automobila u KW !");
      try{
        snaga = (Double.parseDouble(strsnaga))*1.3636;
        JOptionPane.showMessageDialog(null,
                "Vas automobil ima "+Math.round(snaga)+" KS");
      }
      catch(Exception ex){
        System.out.println(ex);
        System.exit(0);
      }
      odgovor = JOptionPane.showConfirmDialog(null,
        "Zelite li jos da konvertujete ?",
        "ODLUKA",
        JOptionPane.YES_NO_OPTION);
    }while(odgovor==0);
  }
}
```







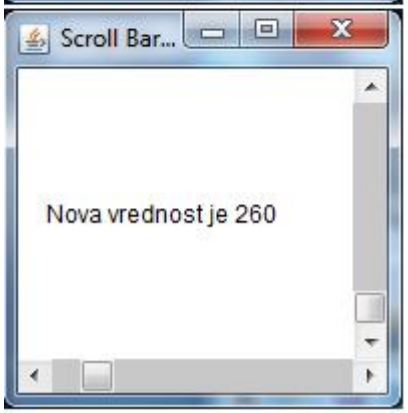
Horizontalni i vertikalni klizači

- U programima se često koriste horizontalni i vertikalni klizači.
- Za implementaciju ovih komponenti koristi se klasa Scrollbar i interfejs AdjustmentListener.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Jsb extends JFrame {
    JLabel jlabel;

    // osluskivac kao unutrasnja klasa
    class MyAdjustmentListener implements AdjustmentListener {
    public void adjustmentValueChanged(AdjustmentEvent e) {
         jlabel.setText("    Nova vrednost je " + e.getValue() +
                        "      ");
        }
    }

    public Jsb(String naslov) {
        setTitle(naslov);
        setSize(200,200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jlabel=new JLabel("pomerite klizac ?");
```

Horizontalni i vertikalni klizači

```
        JScrollBar hbar = new JScrollBar(
                Scrollbar.HORIZONTAL, 30, 20, 0, 300);
                                    //start,korak,min,max
        hbar.setUnitIncrement(2); //mala izmena
        hbar.setBlockIncrement(1);//mala izmena
        hbar.addAdjustmentListener(
             new MyAdjustmentListener());
        JScrollBar vbar = new JScrollBar(
                  Scrollbar.VERTICAL, 30, 40, 0, 300);
        vbar.addAdjustmentListener(
             new MyAdjustmentListener());
        Container c= getContentPane();
        c.setLayout(new BorderLayout());
        c.add(hbar, BorderLayout.SOUTH);
        c.add(vbar, BorderLayout.EAST);
        c.add(jlabel, BorderLayout.CENTER);
        setVisible(true);
    }
    public static void main(String s[]) {
        Jsb j = new Jsb("Scrollbar");
    }
}
```



JScrollPane

- Sledeći primer ilustruje korišćenje klase JScrollPane.
- Kanvas je podeljen na matricu koja ima dva reda i jednu kolonu.
  - U prvom redu se prikazuje slika koja se može skrolovati.

```
import java.awt.*;
import javax.swing.*;

class ScrolledPane extends JFrame
{
   private JScrollPane scrollPane;

   public ScrolledPane()
   {
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("ScrolledPane");
      setSize(300, 200);
      setBackground( Color.gray );
      scrollPane = new JScrollPane();
      Icon image = new ImageIcon("firefox.jpg");
      JLabel label = new JLabel(image);
      scrollPane.getViewport().add(label);
      getContentPane().setLayout(new GridLayout(2,1));
      getContentPane().add(scrollPane);
   }
```

JScrollPane

```
    public static void main( String args[] )
    {
        ScrolledPane mainFrame= new ScrolledPane();
        mainFrame.setVisible(true);
    }
}
```

JTabbedPane

- Korišćenjem klase JTabbedPane kreiraju se tri tab-a:
  - unos korisničkog imena i lozinke (maskirani unos)
  - grupa programskih dugmadi
  - grupa za višelinijski za unos teksta.

  Svakom tab-u se pridružuje panel na koji će se postavljati odgovarajuće navedene komponente.

```
import java.awt.*;
import javax.swing.*;

class TabbedPane extends JFrame {

  private JTabbedPane tabbedPane;
  private JPanel panel1;
  private JPanel panel2;
  private JPanel panel3;

  public TabbedPane()  {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle( "JTabbedPane" );
    setSize( 300, 200 );
    setBackground( Color.gray );
```

JTabbedPane

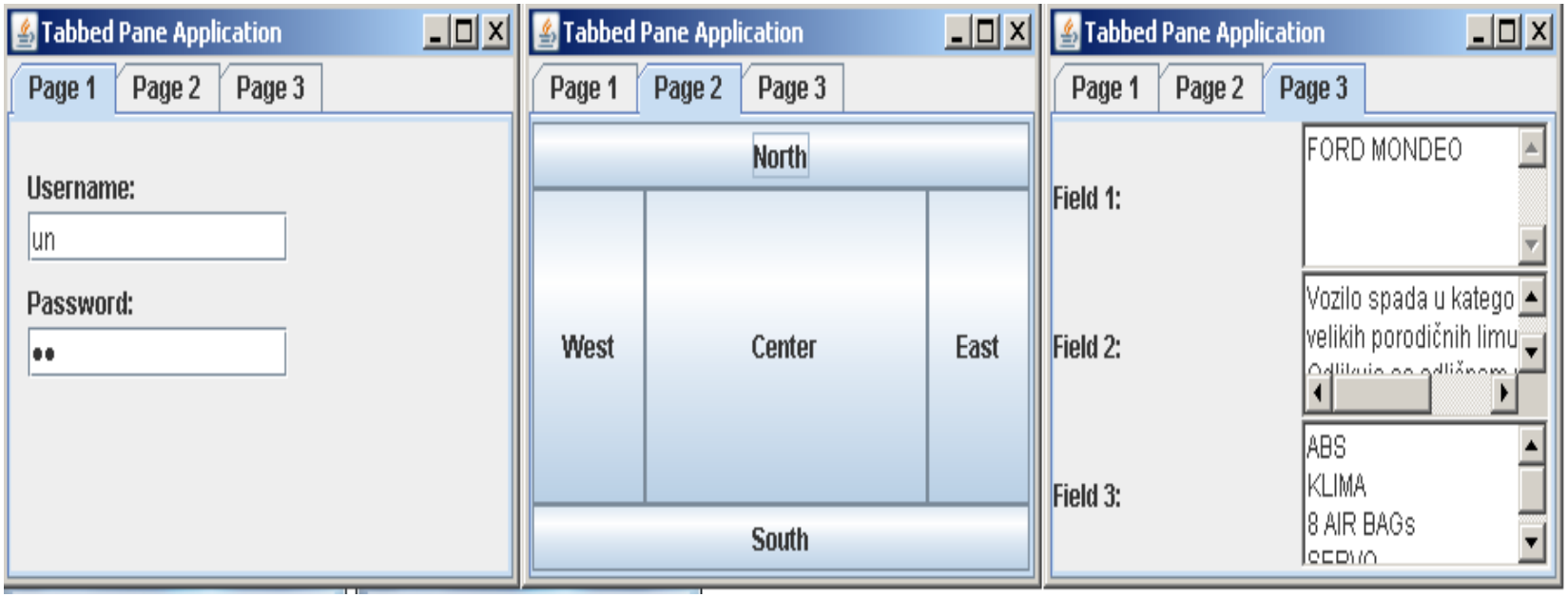
```
    // kreiranje tab-ova
    createPage1();
    createPage2();
    createPage3();
    // kreiranje tabbed pane
    tabbedPane = new JTabbedPane();
    tabbedPane.addTab( "Page 1", panel1 );
    tabbedPane.addTab( "Page 2", panel2 );
    tabbedPane.addTab( "Page 3", panel3 );
    getContentPane().add( tabbedPane, BorderLayout.CENTER );
  }
  public void createPage1()  {
    panel1 = new JPanel();
    panel1.setLayout( null );//apsolutno postavljanje
    JLabel label1 = new JLabel( "Username:" );
    label1.setBounds( 10, 15, 150, 20 );
    panel1.add( label1 );
    JTextField field = new JTextField();
    field.setBounds( 10, 35, 150, 20 );
    panel1.add( field );
    JLabel label2 = new JLabel( "Password:" );
    label2.setBounds( 10, 60, 150, 20 );
    panel1.add( label2 );
```

JTabbedPane

```
    JPasswordField fieldPass = new JPasswordField();
    fieldPass.setBounds( 10, 80, 150, 20 );
    panel1.add( fieldPass );
  }
  public void createPage2()  {
    panel2 = new JPanel();
    panel2.setLayout( new BorderLayout() );
    panel2.add( new JButton("North"), BorderLayout.NORTH );
    panel2.add( new JButton("South"), BorderLayout.SOUTH );
    panel2.add( new JButton("East" ), BorderLayout.EAST );
    panel2.add( new JButton("West" ), BorderLayout.WEST );
    panel2.add( new JButton("Center"), BorderLayout.CENTER );
  }
  public void createPage3()  {
    panel3 = new JPanel();
    panel3.setLayout( new GridLayout( 3, 2 ) );
    panel3.add( new JLabel( "Field 1:" ) );
    panel3.add( new TextArea() );
    panel3.add( new JLabel( "Field 2:" ) );
    panel3.add( new TextArea() );
    panel3.add( new JLabel( "Field 3:" ) );
    panel3.add( new TextArea() );
  }
```

JTabbedPane

```
  public static void main( String args[] )  {
    TabbedPane mainFrame= new TabbedPane();
    mainFrame.setVisible( true );
  }
}
```

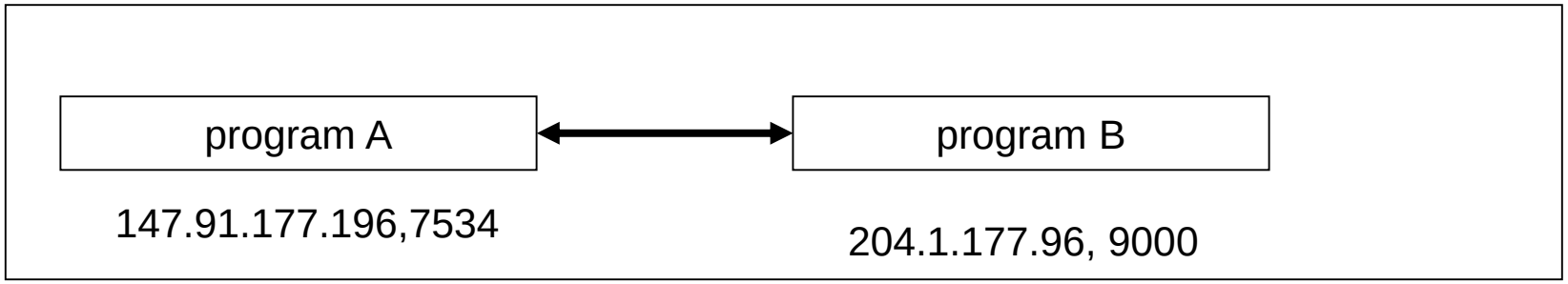


Osnovne karakteristike

- Pod “mrežnim programiranjem” u programskom jeziku Java podrazumeva se pisanje programa koji komuniciraju sa drugim programima preko računarske mreže.
- Zahvaljujući konceptu prenosivog izvršnog koda, pisanje ovakvih programa je istovetno na različitim hardversko/softverskim platformama.
- Komunikacija putem računarske mreže u Java programima podrazumeva korišćenje IP mrežnog protokola.
- Standardna Java biblioteka poseduje klase za komunikaciju preko ovakve mreže korišćenjem TCP i UDP protokola.
- Komuniciranje između dve mašine odvija se putem tokova (*streams*).
- Svaka konekcija između dva programa je dvosmerna za svakog od njih, u smislu da oba programa koji učestvuju u konekciji koriste *stream* za čitanje i *stream* za pisanje.
- *Stream*-ovi se koriste na isti način kao što se koriste prilikom rada sa datotekama u okviru fajl-sistema.
- Klase standardne biblioteke namenjene za pristup mrežnim funkcijama nalaze se u paketu *java.net*, a familija stream klasa koja se takođe koristi nalazi se u paketu *java.io*.

Pojam socket-a

- Za vezu između dva programa na mreži karakterističan je pojam *socket*-a.
- *Socket* predstavlja uređeni par (IP adresa, port) jednog učesnika u komunikaciji.
- Uspostavljena veza između dva programa je zapravo skup dva *socket*-a. Slika 1 ilustruje uspostavljenu vezu između dva programa sa stanovišta *socket*-a.

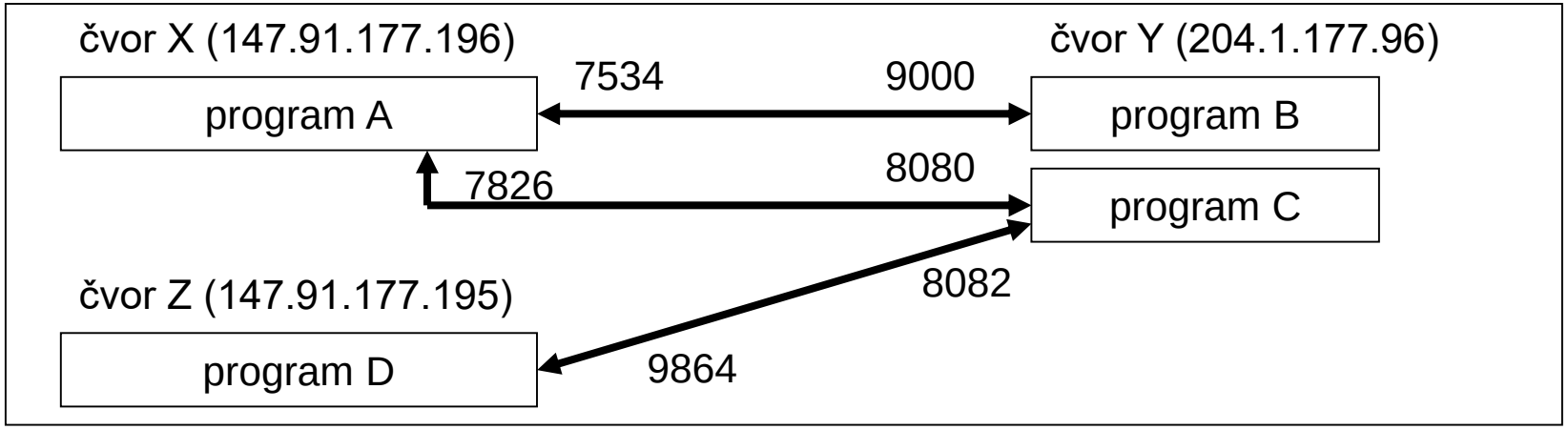


Slika 1. Uspostavljena veza između dva programa

- Kada se govori o vezi, govori se o vezi "između dva programa", a ne o vezi "između dva računara".
- Dva programa koji učestvuju u vezi mogu se izvršavati i na istom računaru.
- Jedan računar može istovremeno izvršavati više programa koji pristupaju mreži.

Pojam socket-a

- Koncept porta je upravo način da se omogući razlikovanje više programa koji su pokrenuti na istom računaru (tj. na istoj IP adresi) i istovremeno pristupaju mreži.
- Slika 4.2 ilustruje situaciju kada se na jednom čvoru mreže izvršavaju dva programa, i jedan od njih ima vezu sa dva programa istovremeno.
- Program A (na čvoru X sa IP adresom 147.91.177.196) ima uspostavljenu vezu sa programom B (na čvoru Y).
- Port koji koristi program A za ovu vezu je 7534, a port koji koristi program B je 9000.
- Program A ima još jednu uspostavljenu vezu, sa programom C, preko svog porta 7826, ka portu 8080 čvora Y.



Slika 2. Slučaj više uspostavljenih veza između programa

Identifikacija čvorova mreže

- Identifikator čvora u IP mreži je IP adresa – 32-bitni broj.
- Ovakve adrese se često, radi lakšeg pamćenja, pišu u formatu koji se sastoji od četiri decimalno zapisana okteta razdvojena tačkom (na primer, 147.91.177.196).
- Java standardna biblioteka poseduje klasu *InetAddress* koja predstavlja IP adresu.
- Kreiranje objekta ove klase se najčešće obavlja pozivom statičke metode *getByName*. Ova metoda prima string parametar koji sadrži bilo IP adresu zapisanu u oktetima, bilo simboličku adresu (npr. *java.sun.com*).

```
InetAddress a = InetAddress.getByName("java.sun.com");
InetAddress b = InetAddress.getByName("147.91.177.196");
```

- Statička metoda *getLocalHost* generiše *InetAddress* objekat koji predstavlja adresu mašine na kojoj se program izvršava:

```
InetAddress c = InetAddress.getLocalHost();
```

Klasa Socket

- Objekti klase *java.net.Socket* predstavljaju uspostavljene TCP konekcije.
- Prilikom kreiranja objekta klase *Socket* vrši se uspostavljanje veze.
- Otvaranje konekcije vrši na jedan od sledećih načina:

```
Socket s1 = new Socket(addr, 25);// addr je InetAddress objekat
Socket s2 = new Socket("java.sun.com", 80);
```

- Kreiranje *Socket* objekta, tj. otvaranje konekcije, omogućuje da se preuzmu reference na *stream* objekte koji se koriste za slanje i primanje poruka.

  Primer inicijalizacije:

```
// inicijalizacija ulaznog streama
BufferedReader in =
  new BufferedReader(
    new InputStreamReader(
      sock.getInputStream()));
// inicijalizuj izlazni stream
PrintWriter out =
  new PrintWriter(
    new BufferedWriter(
      new OutputStreamWriter(
        sock.getOutputStream())), true);
```

Klasa Socket

- Odgovarajući *Reader/Writer* objekti generišu se na osnovu *stream*-ova koje obezbeđuje *Socket* objekat, metodama *getInputStream* i *getOutputStream*.
- Komunikacija sa programom sa kojim je uspostavljena konekcija sada se može odvijati putem poziva odgovarajućih metoda *Reader* i *Writer* klasa.

  Primer:

```
out.writeln("Hello");
String response = in.readLine();
```

- Prekid komunikacije treba završiti propisnim zatvaranjem konekcije. Zatvaranje konekcije se nejčešće svodi na zatvaranje ulaznog i izlaznog *stream*-a i zatvaranje *socket*-a.

  Primer:

```
out.close();
in.close();
sock.close();
```

Tipičan tok komunikacije – klijent strana

- Uloga klijenta u klijent/server komunikaciji podrazumeva nekoliko stvari:
  - Klijent inicira komunikaciju.
  - Nakon uspostavljanja veze, komunikacija se obično svodi na niz parova zahtev/odgovor poruka. Zahteve šalje klijent, a odgovore server.
  - Klijent prekida komunikaciju.
- Ovakva sekvenca aktivnosti može biti predstavljena sledećim segmentom programa:

```
// inicijalizacija
Socket s = new Socket(addr, port);
BufferedReader in = new BufferedReader(... s.getInputStream());
PrintWriter out = new PrintWriter(... s.getOutputStream())), true);
// komunikacija
out.println("zahtev");              // šaljem zahtev
String response = in.readLine();  //  čitam odgovor
     // i tako potreban broj puta...
// prekid veze
in.close();
out.close();
s.close();
```

Klasa ServerSocket

- Klasa *java.net.ServerSocket* koristi se na serverskoj strani. Glavna metoda u ovoj klasi je *accept* – metoda koja blokira izvršavanje programa sve dok neki klijent ne uspostavi vezu na portu na kome *ServerSocket* očekuje klijente.
- Objekti klase *ServerSocket* kreiraju se na standardan način, operatorom **new**.
- Parametar konstruktora je port na kome će server očekivati klijente; kao IP adresa se podrazumeva IP adresa lokalne mašine.

  Primer:

```
ServerSocket ss = new ServerSocket(9000);
```

- Ovim je konstruisan *ServerSocket* objekat pomoću koga će se očekivati klijenti na portu 9000. Samo "osluškivanje" na datom portu inicira se pozivom metode *accept*.
- Ova metoda blokira izvršavanje programa sve dok neki klijent ne uspostavi vezu.
- Rezultat metode je inicijalizovani *Socket* objekat koga serverski program dalje koristi za komunikaciju sa klijentom koji je uspostavio vezu.
- Poziv *accept* metode izgleda ovako:

```
Socket s = ss.accept();
```

Tipičan tok komunikacije – server strana

- Tipičan scenario ponašanja serverskog programa je sledeći:
  - Konstrukcija *ServerSocket* objekta.
  - Očekivanje klijenta metodom *accept*.
  - Komunikacija sa klijentom:
    - Inicijalizacija *stream*-ova
    - Komuniciranje po principu prijem zahteva/slanje odgovora.
    - Završavanje komunikacije – oslobađanje resursa.
- Ovakav scenario može se predstaviti sledećim segmentom programa:

```
// čeka klijenta...
ServerSocket ss = new ServerSocket(port);
Socket s = ss.accept();
// inicijalizacija
BufferedReader in = new BufferedReader(... s.getInputStream());
PrintWriter out = new PrintWriter(... s.getOutputStream())), true);
// komunikacija
String request = in.readLine();    // čita zahtev
out.println("odgovor");            // šalje odgovor
// prekid veze
in.close();
out.close();
s.close();
```

Server koji opslužuje više klijenata

- Prethodni primer je prikazao serverski program koji komunicira sa jednim klijentom – nakon što ga server sačeka, komunikacija između klijenta i servera se obavi i potom završi.
- Ovakvi serverski programi su vrlo retki – serveri se konstruišu tako da mogu da opslužuju više klijenata i to istovremeno.
- Potreba da server komunicira sa više klijenata istovremeno se može rešiti uvođenjem posebnih programskih niti za komunikaciju sa klijentima, tako da se sa svakim klijentom komunikacija obavlja u posebnoj programskoj niti.
- Za *n* istovremenih klijenata postojaće *n* ovakvih programskih niti. Sa stanovišta implementacije u programskom jeziku Java, ove niti predstavljene su odgovarajućom klasom koja nasleđuje klasu *Thread*.

```
// obrada pojedinačnog zahteva
class ServerThread extends Thread {
  public void run() {
    // inicijalizacija
    // komunikacija
    // prekid veze
  }
}
```

Server koji opslužuje više klijenata

- Pored niti za komunikaciju sa pojedinim klijentima, potrebna je i posebna nit koja "osluškuje" serverski port.
- Nakon uspostavljanja veze, serverska nit pokreće nit za komunikaciju sa klijentom, a sama se vraća u stanje čekanja na novog klijenta.
- Program na serverskoj strani se sastoji od *n+1* niti prilikom obrade *n* istovremenih klijentskih zahteva.

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

Primer klijent/server komunikacije

Primer: konstruisati jednostavnu klijent/server aplikaciju, gde su dati sledeći zadaci klijenta:

- Uspostavlja vezu sa serverom.
- Šalje zahtev serveru (tekst “HELLO”).
- Čita odgovor servera.
- Ispisuje odgovor na konzolu.
- Završava komunikaciju.

- Klijentski program je predstavljen klasom *Client1*.

```
package tcpip;
import java.io.*; import java.net.*;
public class Client1 {
  public static final int TCP_PORT = 9000;
  public static void main(String[] args) {
    try {
      // odredi adresu racunara sa kojim se povezujemo (ovde lokalno)
      InetAddress addr = InetAddress.getByName("127.0.0.1");
      // otvori socket prema serveru
      Socket sock = new Socket(addr, TCP_PORT);
      // inicijalizuj ulazni stream
      BufferedReader in = new BufferedReader(
                          new InputStreamReader(sock.getInputStream()));
```

Primer klijent/server komunikacije

```
        // inicijalizuj izlazni stream
        PrintWriter out =
            new PrintWriter(
              new BufferedWriter(
                new OutputStreamWriter(
                  sock.getOutputStream())), true);
        // posalji zahtev
        System.out.println("[Client]: HELLO");
        out.println("HELLO");
        // procitaj odgovor
        String response = in.readLine();
        System.out.println("[Server]: " + response);
        // zatvori konekciju
        in.close();
        out.close();
        sock.close();
      } catch (UnknownHostException e1) {
          e1.printStackTrace();
      } catch (IOException e2) {
          e2.printStackTrace();
      }
    }
}
```

Primer klijent/server komunikacije

- Zadatak serverskog programa je sledeći:
  - Čeka klijente u beskonačnoj petlji.
  - Za svakog klijenta koji je uspostavio vezu pokreće posebnu nit koja radi sledeće:
    - Čita zahtev klijenta (tekst “HELLO”).
    - Šalje odgovor – redni broj obrađenog zahteva.
- Osnovna nit servera u kojoj se očekuju klijenti nalazi se u klasi *Server1*:

```
package tcpip;
import java.io.*; import java.net.*;
public class Server1 {
  public static final int TCP_PORT = 9000;
  public static void main(String[] args) {
    try {
      int clientCounter = 0;
      ServerSocket ss = new ServerSocket(TCP_PORT);
      System.out.println("Server running...");
      while (true) {
        Socket sock = ss.accept();
        System.out.println("Client accepted:"+ (++clientCounter));
        ServerThread1 st = new ServerThread1(sock, clientCounter);
      }
    } catch (Exception ex) {  ex.printStackTrace(); }  } }
```

Primer klijent/server komunikacije

- U okviru osnovne niti nalazi se beskonačna while petlja u okviru koje se očekuju klijenti i pokreće nit za komunikaciju sa klijentom.
- Konstruktor ove niti prima kao argumente *Socket* objekat koji će koristiti u komunikaciji (*sock*) i redni broj klijenta koji se prijavio (*clientCounter*).
- Nit za komunikaciju predstavljena je klasom *ServerThread1*:

```
package tcpip;
import java.io.*;import java.net.*;
public class ServerThread1 extends Thread {
 public ServerThread1(Socket sock, int value) {
    this.sock = sock;
    this.value = value;
    try {
      in = new BufferedReader(           // inicijalizuj ulazni stream
             new InputStreamReader(
               sock.getInputStream()));
      out = new PrintWriter(             // inicijalizuj izlazni stream
              new BufferedWriter(
                new OutputStreamWriter(
                  sock.getOutputStream())), true);
    } catch (Exception ex) { ex.printStackTrace(); }
    start();
}
```

Primer klijent/server komunikacije

```
  public void run() {
    try {
      // procitaj zahtev
      String request = in.readLine();
      // odgovori na zahtev
      out.println("(" + value + ")");
      // zatvori konekciju
      in.close();
      out.close();
      sock.close();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
  private Socket sock;
  private int value;
  private BufferedReader in;
  private PrintWriter out;
}
```

Primer klijent/server komunikacije

- Komunikacija između Web servera i klijenta (Web čitača), podseća na komunikaciju prikazanu u prethodnom primeru.
- Zahtev Web čitača tipično sadrži naziv datoteke koju čitač traži.
- Odgovor servera je poruka u kojoj se nalazi tražena datoteka.
- U prethodnom primeru *format poruka* koje se razmenjuju između klijenta i servera bio je:
  - zahtev klijenta je tekst **HELLO** koji se završava znakom za novi red (*linefeed*, LF).
  - Slanje ovakve poruke postiže se sledećim pozivom u okviru klijentskog programa

```
out.println("HELLO");
```

  - server očitava zahtev klijenta pomoću poziva metode readLine:

```
String request = in.readLine();
```

    Ova metoda će blokirati izvršavanje programa sve dok se na ulazu ne pojavi znak za novi red (LF) i tada će vratiti tekst koji je sa mreže pristigao pre tog znaka.

Primer klijent/server komunikacije

- Korišćenje znaka LF kao oznake kraja poruke (ili kraja jednog dela poruke) je relativno često u specifikaciji protokola.
- Sa druge strane, nije obavezno koristiti baš LF kao oznaku kraja poruke.
- Komunikacioni protokol može biti tako specificiran da je dužina poruke koja se očekuje unapred poznata, tako da takvu poruku možemo pročitati pozivom

```
in.read(buffer, 0, length);
```

- U slučaju da dužina poruke nije unapred poznata, korišćenje karaktera LF je zgodno jer postoji metoda *readLine* koja blokira izvršavanje sve dok taj karakter ne pristigne sa mreže.
- U slučaju da je odluka da se ne koristi karakter LF nego neki drugi, mora da se implementira funkcionalnost ove metode.

Primer - klijent i server za listanje sadržaja direktorijuma

- Napisati klijent/server aplikaciju koja omogućava listanje sadržaja direktorijuma sa servera na klijentu.
- Na klijentov zahtev koji sadrži putanju direktorijuma na serveru koga treba izlistati, server formira spisak i vraća ga klijentu kao odgovor.
- Klijent i server nemaju potrebe za GUI interfejsom.
- *Komentar*: klijent i server će izgledati nalik klijentu i serveru koji su prikazani u prethodnom primeru.
- Ono što je neophodno uraditi kao prvo, je definisati protokol komunikacije klijenta i servera, pre svega format poruka koje se šalju tokom komunikacije.
- Za listanje sadržaja direktorijuma treba pogledati kako se koristi klasa *java.io.File* i njene metode
  - *exists*
  - *isDirectory*
  - *listFiles*.
- Odgovor servera bi trebalo da na odgovarajući način reaguje na situacije kada traženi direktorijum ne postoji, ili kada je u pitanju fajl, a ne direktorijum.

Server2.java

```
package tcpip;
import java.io.*;
import java.net.*;
public class Server2 {
  public static final int TCP_PORT = 9000;
  public static void main(String[] args) {
    try {
      int clientCounter = 0;
      // slušaj zahteve na datom portu
      ServerSocket ss = new ServerSocket(TCP_PORT);
      System.out.println("Server running...");
      while (true) {
        Socket sock = ss.accept();
        System.out.println("Client accepted: " + (++clientCounter));
        ServerThread2 st = new ServerThread2(sock, clientCounter);
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

ServerThread2.java

```
package tcpip;
import java.io.*; import java.net.*;
public class ServerThread2 extends Thread {
  public ServerThread2(Socket sock, int value) {
    this.sock = sock;
    this.value = value;
    try {
      // inicijalizuj ulazni stream
      in = new BufferedReader(
             new InputStreamReader(
               sock.getInputStream()));
      // inicijalizuj izlazni stream
      out = new PrintWriter(
              new BufferedWriter(
                new OutputStreamWriter(
                  sock.getOutputStream())), true);
      // pokreni thread
      start();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
```

ServerThread2.java

```
public void run() {
    String response = "";
    try {
      String request = in.readLine(); // procitaj zahtev
      File file = new File(request);  // formiraj odgovor
      if (file.exists()) {
        if (file.isDirectory()) {
          File[] files = file.listFiles();
          for (int i = 0; i < files.length; i++)
            response += files[i].getName() + "\n";
        } else {   response = "Error: " + request + " is a file\n"; }
      } else {     response = "Error: path does not exist\n";       }
      response += "END";       // pošalji odgovor
      out.println(response);  // zatvori konekciju
      in.close();      out.close();      sock.close();
    } catch (Exception ex) { ex.printStackTrace();                   }
  }
  private Socket sock;
  private int value;
  private BufferedReader in;
  private PrintWriter out;
}
```

Client2.java

```
package tcpip;
import java.io.*;
import java.net.*;
public class Client2 {
  public static final int TCP_PORT = 9000;
  public static void main(String[] args) {
  // direktorijum i adresa servera se zadaju iz komandne linije
  // ako nisu definisani, ispiši poruku i prekini sa radom
  if (args.length == 0) {
    System.out.println("Remote Directory Client v1.0");
    System.out.println("Usage: Client2 <full-dir-path> [<hostname>]");
    System.out.println("Parameters:");
    System.out.println("  <full-dir-path>  The full directory pathname");
    System.out.println(
      "  <hostname> Server name; default is localhost\n\n");
    System.exit(0);
  }
  String path = args[0];
  String hostname = (args.length > 1) ? args[1] : "localhost";
```

Client2.java

```
    try {
      InetAddress addr = InetAddress.getByName(hostname);
      Socket sock = new Socket(addr, TCP_PORT);
      BufferedReader in = new BufferedReader(new
                              InputStreamReader(sock.getInputStream()));
      PrintWriter out = new PrintWriter(new BufferedWriter(new
                    OutputStreamWriter(sock.getOutputStream())), true);
      System.out.println("Querying server...");
      out.println(path);
      String response;
      String list = "";
      while (!(response = in.readLine()).equals("END")) {
        list += response + "\n";
      }
      System.out.println("\nServer responded:\n" + list);
      in.close();
      out.close();
      sock.close();
    } catch (UnknownHostException e1) { e1.printStackTrace();
    } catch (IOException e2) { e2.printStackTrace(); }
  }
}
```