PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI203 - JAVA 6: NAPREDNO PROGRAMIRANJE U JAVI

Programiranje u mreži

Lekcija 02

PRIRUČNIK ZA STUDENTE

KI203 - JAVA 6: NAPREDNO PROGRAMIRANJE U JAVI

Lekcija 02

PROGRAMIRANJE U MREŽI

- Programiranje u mreži
- Poglavlje 1: Umrežavane aplikacije
- Poglavlje 2: Kilijent-server povezivanje računara
- Poglavlje 3: Klasa InetAddress
- Poglavlje 4: Studija slučaja: Distribuirane igre Tic-Tac-Toe
- Poglavlje 5: Primeri:JSoup, JavaMail biblioteka i Process klasa
- Poglavlje 6: Domaći zadatak
- Zaključak

UVOD

Ciljevi lekcije

Ova lekcija ima za cilj da:

1. objasni termine: TCP, IP, ime domena, ime domenskog servera, komunikacije sa tokovima, i komunikacije sa paketima;
2. kreira servere upotrebom serveskih soketa i klejente, upotrebom klijentskih soketa;
3. primeni Javin program umrežavanja upotrebom strim soketa;
4. razvije jedan primer klijent-server aplikacije;
5. pokaže dobijanje Internet adresa primenom InetAddress klase;
6. razvije servere sa više klijenata;
7. šalje i prima objekte preko mreže;
8. razvije interaktivnu aplikaciju igre tic-tac-toe za igru preko Interneta.

Navedeni ciljevi koji čine i program ove lekcije, se detaljnije izučava u više drugih predmeta (a najviše na predmetu CS230 Distribuirani sistemi). Ova materija se ovde izlaže da bi studenti prve godine mogli da razumeju osnovne principe izrade programa koji rade u mreži (misli se na računarske mreže

Referenca: Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste,

Poglavlje 1

Umrežavane aplikacije

PROTOKOLI UMREŽAVANJA RAČUNARA

IP je protokol niskog nivoa koji služi za isporuku podataka sa jednog računara na drugi. TCP omogućava da dva servera razmene tokove (strimove) podataka, a UDP – paketa podataka.

Mnoge aplikacije u svom radu koriste više računara povezanih preko interneta (koristeći DSL ili kablovske modeme) ili preko lokalne računarske mreže (Local Area Network–LAN). Da bi dva računara mogli da komuniciraju, moraju prvo da znaju uzajamne adrese, sa kojim su registrovani na Internetu. Internet protokol (Internet Protocol– IP) je adresa koja na jednistveni način utvrđuje računar na Internetu. IP adresa je sačinjena od četiri broja od 0 do 255 između kojih se nalaze tačke, kao što je na primer. 130.254.204.31. Da bi se lakše pamtili, njima se mogu pridodavati imena koja se lakše pamte, i ona imaju naziv: imena domena (domain names), kao što je na primer: metropolitan.ac.rs . Specijalni serveri se nazivaju serveri sa imenima domena (Domain Name Servers - DNS) koji se nalaze na Internetu i prevode imena servera (tj. njihova imena domena) u IP adrese. Kada neki računar želi da stupi u kontakt sa računarom sa imenom metropolitan.ac.rs , on mora prvo da pita DNS da prevede domensko ime računara u njegovu brojčanu IP adresu i da onda pošalje zahtev upotrebom IP adrese.

Internet Protocol (IP) je protokol niskog nivoa koji služi za isporuku podataka sa jednog .računara na drugi. Dva protokola višeg nivoa su Transmission Control Protocol (TCP), i User Datagram Protocol (UDP).

TCP omogućava da dva servera uspostave vezu i da razmene tokove podataka. TCP garanuje isporuku podataka podataka i da će paketi podataka biti isporučeni u istom redosledu po kom su poslati.

UDP je standardni protokol veze računara sa računarom, koji se koristi IP, a koji omogućava aplikaciji da pošalje datagram iz aplikacije sa jednog računara na drugi.

Java podržava komunikacije i sa tokovima podataka i sa paketima. Komunikacije sa tokovima upotrebljavaju TCP za prenos podataka, dok komunikacije sa paketimi upotrebljavaju UDP.

Kako TCP otkriva izgubljene podatke u transmisiji te ih ponovo šalje, on je pouzdan jer garantuje prenos svih podataka, bez gubitaka. S druge strane, *UDP, ne može da garantuje da neće doći do gubljenja podataka.*

Komunikacije sa tokovima podataka se koriste kod programiranja u Javi. Komunikacije sa paketima podataka je opisana u vidu dodatne dokumentacije definisane u *Supplement III.P, Networking Using Datagram Protocol.*

1.1 Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD STUDENATA

Provežbati stečena znanja

Zadatak 1:

Pronaći IP adresu vašeg računara. Zbog čega je bitno koristiit IP adresu? Čemu ona služi?

Poglavlje 2

Kilijent-server povezivanje računara

UTIČNICE NA SERVERU I KLIJENTU

Java obezbeđuje klase ServerSocket i Socket za kreiranje serverske i klijentske softverske utičnice. Razmena U/I tokova podataka između računara zahteva serverske i klijentske utičnice.

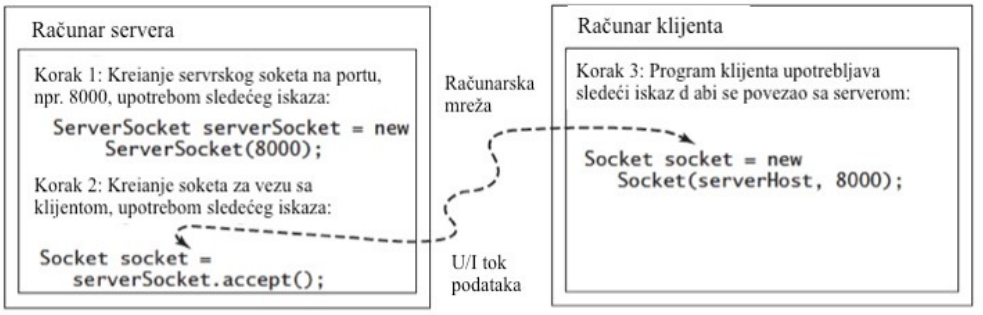
Dva programa komuniciraje preko Interneta preko serverske i klijentske softverske utičnice (sockets) upotrebom ulazno-izlaznih (U/I) tokova podataka (streams).

Umrežavanje računara je jako integrisano u Javi. Java API obezbeđuje klase za kreiranje utičnica (sockets) koji olakšavaju kominikacije programa preko Interneta. **Utičnice** su krajnje tačke logičkih veza između dva računara i koriste se za slanje i primanje podataka. Java tretira komunikacije preko soketa kao da su to niti U/I operacija. Na taj način, program može da uzima podatke sa utičnice ili da šalje podatke utičnici, isto tako lako kao da čitaju neku datoteku ili zapsiju podatke u datoteci.

.Programiranje mrežnih aplikacija obično obuhvata jedan server i jednog ili više klijenta. Klijenti šalju zahteve serveru, a server im odgovara..

Klijent počinje sa pokušajem da prvo uspostavi vezu sa serverom. Server može da prihvati ili da odbije taj zahtev.

Kada je veza uspostavlena *, klijent i server komuniciraju preko softverskih utičnica* , tj. objekata klasa***ServerSocket*** i ***Socket***. Server mora da radi u vreme kada klijent pokuša da uspostavi vezu sa serverom. Server ćeka zahtev za uspostavljanj eveze od klijenta. Potrebne su programske naredbe za kreiranje soketa na serveru i klijentu (slika 1).



Slika 2.1.1 Server kreira svoj soket, a kada uspostavi vezu sa klijentom, povezuje se sa soketom klijenta

UTIČNICE SERVERA

Utičnica servera obezbeđuje vezu sa utičnicom klijenta, a pri njihovom kreiranju treba da definišete broj utikača na serveru koji će utičnica koristiti.

Da bi ste uspostavili neki server, prvo morate da kreirate softversku utičnicu (soket) servera i da ga dodelite nekom njegovom priključku, tj. komponenti računara koja služi za vezu računara sa drugim uređajima.Priključak (port) je podržan odgovarajućim softverskim osluškivačima njegovih veza. Priključak (port) utvrđuje TCP servis koji je definisan u utičnici. Broj priključka se kreće od 0 do 65536, ali su brojevi od 0 do 1024 rezervisani za privilegovane servise. Na primer, server elektronske pošte komunicira preko priključka sa brojem 25, a veb server obično koristi broj 80. Vi možete da izaberete bilo koji broj priključka koji trenutno nije u upotrebi od strane drugih programa.

Sledeći izkaz kreira jedan soket servera:

:

```
ServerSocket serverSocket = new ServerSocket(port);
```

UTIČNICE KLIJENTA

Utičnica na računaru klijenta je neophodna radi komunikacije sa serverom, naznačenim imenom ili IP adresom, preko njegovog porta, označenim njegovim brojem.

Posle kreiranja softverske utičnice (soketa) na serveru, server može da koristi sledeći iskaz za služanje veza, tj. za primanje podataka preko veza računara:

```
Socket socket = serverSocket.accept();
```

Ovaj iskaz čeka dok se klijent ne poveže sa utičnicom servera. Klijent koristi sledeći iskaz za podnošenje zahteva za uspostavljanje veze sa serverom:

```
Socket socket = new Socket(serverName, port);
```

Ovaj iskaz otvara utičnicu, preko navedenog priključka, tako da program klijenta može da sada komunicira sa serverom. Server koristi ime definisano promenljivom **serverName**, čija vrednost je iili Internet ime domen servera, ili njegova IP adresa. Na primer, sledeći iskaz kreira utičnicu na računaru klijenta da bi ga povezao sa računarom servera sa IP adresom 130.254.204.33 preko porta 8000:

```
Socket socket = new Socket("130.254.204.33", 8000)
```

Umesto brojčane IP adrese, možete koristiti i domensko ime servera, kao što pokazuje sledeći primer:

```
Socket socket = new Socket("liang.armstrong.edu", 8000);
```

Kada kreirate softversku utičnicu sa domenskim imenom njegovog serevra, JVM pita da DSN da bi mogla da prevede domensko ime u brojčanu IP adresu.

Napomena: Program može da koristi i naziv servera ***localhost*** ili IP adresu 127.0.0.1 koja se odnosi na računar na kome klijent radi.

PRENOS PODATAKA PREKO UTIČNICA

Metodi getInputStream() i getOutputStream() na priključnici tipa Socket ili ServerSocket kreiraju ulazni, odn. izlazni tok podataka.

Kada server prihvati vezu, komunikacija između servera i klijenta se vodi na isti način kao kod U/I tokova podataka (strimova). Operacije (metodi) koje su potrebne za kreiranje tokova podataka za razmenu podatka su pokazani na slici 2.

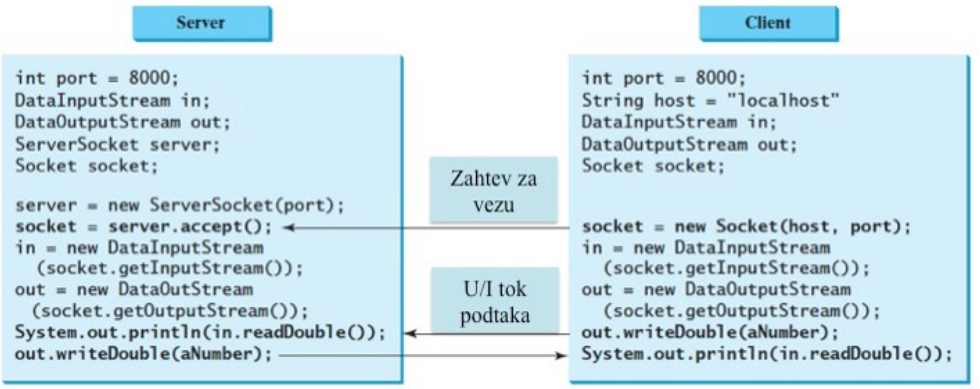
Da bi se dobio ulazni tok podataka i izlazni tok podataka, treba upotrebiti metod ***getInputStream()*** i ***getOutputStream()*** na prikljunici, tj. objektu klase Socket ili ServerSocket..

```
InputStream input = socket.getInputStream();
 OutputStream output = socket.getOutputStream();
```

Tokovi podataka InputStream i OutputStream unose i iznose bajtove. Možete ***koristi DataInputStream, DataOutputStream, BufferedReader*** i ***PrintWriter*** da unosite ili iznosite podatke, kao što su ***int, double***, ili ***String***.

Sledeći Java iskazi, na primer, kreiraju ***DataInputStream*** ulazni tok podataka, i ***DataOutput Stream*** izlazni tok podataka, da bi uneli, odn. Izneli podatke primitivnog tipa.

```
DataInputStream input = new DataInputStream(socket.getInputStream());
 DataOutputStream output = new DataOutputStream(socket.getOutputStream());
```



Slika 2.1.2 Server i klijent razmenjuju podatke koristeći U/I tokove podataka pored i utičnica (soketa)

Server upotrebljava metod ***input.readDouble()*** da primi podatke tipa***double*** od klijenta i šalje klijentu podatke tipa ***double*** metodom ***writeDouble()***.

VIDEO: JAVA SOKETI (UTIČNICE)

Java Sockets Tutorial (9,51 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: UVOD U UTIČNICE JAVE

Java - Sockets - Introduction - 1 of 3 (12,26 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PROGRAMIRANJE UTIČNICA

Socket Programming in Java (2,41 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PROGRAMIRANJE UTIČNICA - KLIJENT

Socket Programming in Java One Way (12,00 minuta)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO: PROGRAMIRANJE UTIČNICA - SA SERVEROM

Socket Programming in Java Two Way (4,58 minuta)

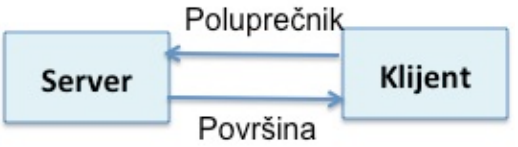
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

2.1 Pokazni primeri

PRIMER 1: OPIS PROBLEMA

Klijent šalje podatke serveru. Server prima podatke, upotrebljava ih da bi proizveo rezultat, i onda šalje nazad rezultat klijentu

Primer predstavlja i klijentski i serverski program. Klijent šalje podatke serveru. Server prima podatke, upotrebljava ih da bi proizveo rezultat, i onda šalje nazad rezultat klijentu. Klijent prikazuje rezultat na svojoj konzoli (monitoru). Podaci koji se šalju predstavljaju poluprelnik kruga a rezultat koji proizvodi servere je površina kruga (slika 1)

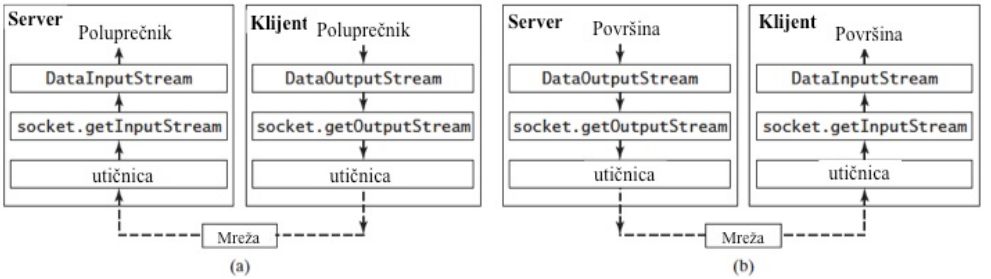


Slika 2.2.1 Opis problema

Textbox

Klijent šalje poluprečnik koristeći izlazni tok podataka DataOutputStream na utičnici izlaznog toka podataka, server prima poluprečnik preko ulaznog tok podataka DataInputStream na utičnici ulaznog toka podataka, kao što je pokazano na slici 2a.

Server računa površinu kruga i šalje je klijentu preko izlaznog toka podataka DataOutputStream na utičnici izlaznog toka podataka. Klijent prima površinu kruga preko ulaznog toka podataka DataInputStream na utičnici ulaznog toka podataka, ko što je pokazano na slici 2b.

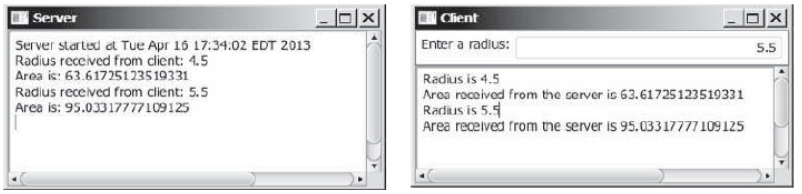


Slika 2.2.2 a) Klijent šalje poluprečnik serveru b) Server šalje poluprečnik klijentu

PRIMER 1: SERVERSKI PROGRAM PRIMERA

Klasa Server kreira objekat ServerSocket.serverSocket i pridodaje ga priključku 8000 upotrebom iskaza

Na slici 2 prikazuje se način unosa poluprečnika u klijentu i prikaza rezultata koje daje server.



Slika 2.2.3 Unos prečnika i prikaz rezultata - površine kruga

Prvo startujete serverski, pa onda i klijentski program, u kome unosite poluprečnik u tekstualno polje, a onda pritiskom na Enter klijent ga šalje serveru. Server računa površinu i vraća je klijentu. Proces se ponavlja sve dok jedan od programa ne završi. Klase umreženja su date u paketu java.net. Unosite taj paket kada pišete mrežne programe u Javi. Klasa Server kreiara objekat ServerSocket.serverSocket i pridodaje ga priključku 8000 upotrebom iskaza (linija 26 u programu Server):

```
ServerSocket serverSocket = new ServerSocket(8000);
```

Sada server ulazi u status čekanja zahteva sa vezu, upotrebom sledećeg iskaza (linija 31 u Server programu:

```
Socket socket = serverSocket.accept();
```

Listing serverskog programa:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package clientserver;

import java.io.*;
import java.net.*;
import java.util.Date;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.stage.Stage;

/**
```

```
 *
 * @author Jovana
 */
public class Server extends Application {

    @Override // Redefinisanje metoda start() klase Application
    public void start(Stage primaryStage) {
        // Prostor za tekst za prikaz sadržaja
        TextArea ta = new TextArea();

        // Kreiranje scene i njeno postavljanje na pozornicu
        Scene scene = new Scene(new ScrollPane(ta), 450, 200);
        primaryStage.setTitle("Server"); // Unos naziva pozornice
        primaryStage.setScene(scene); // Postavljanje scene na pozornicu
        primaryStage.show(); // Prikaz pozornice

        new Thread(() -> {
            try {
                // Kreiranje utičnice servera
                ServerSocket serverSocket = new ServerSocket(8000);
                Platform.runLater(()
                        -> ta.appendText("Server started at " + new Date() + '\n'));

                // Očekivanje zahteva za poveziivanje
                Socket socket = serverSocket.accept();

                // Kreiranje ulaznog i izlaznog toka podataka
                DataInputStream inputFromClient = new DataInputStream(
                        socket.getInputStream());
                DataOutputStream outputToClient = new DataOutputStream(
                        socket.getOutputStream());

                while (true) {
                    // Dobijanje poluprečnika od klijenta
                    double radius = inputFromClient.readDouble();

                    // Proračun površine
                    double area = radius * radius * Math.PI;

                    // Slanje površine nazad klijentu
                    outputToClient.writeDouble(area);

                    Platform.runLater(() -> {
                        ta.appendText("Radius received from client: "
                                + radius + '\n');
                        ta.appendText("Area is: " + area + '\n');
                    });
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }).start();
    }
```

```
    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

PRIMER 1: KLIJENTSKI PROGRAM PRIMERA

Ako se klijent i server izvršavaju na različitim računarima, umesto "localhost" treba staviti IP adresu servera.

Ovde se prikazuje listing klijentskog programa (klasa Client).

Server čeka dok klijent ne pošalje zahtev za uspostavljanj e veze. Posle povezivanja, server dobija poluprečnik od klajenta preko ulaznog toka podataka, računa površinu i šalje rzultat klijentu sa izlaznim tokom podataka. Metod ***ServerSocket accept()*** zahteva vreme za izvršenje. Nije prikladno da se ovaj metod izvršava u niti JavaFX aplikacie. Zato, postavljamo ga u posebnu nit (linija 23-50). Iskazi za osvežavanje GUI se izvršava iz niti JavaFX aplikacije upotrebom ***Platform.runLater()*** metoda (linije 27-28, 49-53).

Klasa Client upotrebljava sledeći iskaz za kreiranje utičnice koja će zahtevati vezu sa serverom na istom računaru (localhost) na priključku 8000 (linija 67 u listingu klase Client.

```
Socket socket = new Socket("localhost", 8000);
```

Ako se klijent i server izvršavaju na različitim računarima, umesto "localhost" stavite IP adresu servera. Ako server ne radi, klijent program završava sa***java.net.ConnectException***. Kada je povezan, klijent se povezuje i klijent dobija ulaz i izlaz – umotan podacima ulaznog i izlaznog toka. – da bi primio i poslao podatak serveru. Ako se javi***java.net.BindException*** pri startovanju servera, to znači da je priključak trenutno zauzet,

Listing klijentskog programa:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package clientserver;

import java.io.*;
import java.net.*;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Client extends Application {

    // IO streams
    DataOutputStream toServer = null;
    DataInputStream fromServer = null;

    @Override // Redefinisanje metoda start() u klasi Application
    public void start(Stage primaryStage) {
        // Panel p za natpis i  tekstualno polje
        BorderPane paneForTextField = new BorderPane();
        paneForTextField.setPadding(new Insets(5, 5, 5, 5));
        paneForTextField.setStyle("-fx-border-color: green");
        paneForTextField.setLeft(new Label("Enter a radius: "));

        TextField tf = new TextField();
        tf.setAlignment(Pos.BOTTOM_RIGHT);
        paneForTextField.setCenter(tf);

        BorderPane mainPane = new BorderPane();
        // Tekstualno polje za prikaz sadržaja
        TextArea ta = new TextArea();
        mainPane.setCenter(new ScrollPane(ta));
        mainPane.setTop(paneForTextField);

        // Kreiranje scene i njeno postavljanje na pozornicu
        Scene scene = new Scene(mainPane, 450, 200);
        primaryStage.setTitle("Client"); // Unos naziva pozornice
        primaryStage.setScene(scene); // Postavljanje scene na pozornicu
        primaryStage.show(); // Prikaz pozornice

        tf.setOnAction(e -> {
            try {
                // Dobijanje poluprečnika iz tekstualnog polja
                double radius = Double.parseDouble(tf.getText().trim());

                // Slanje poluprečnika serveru
                toServer.writeDouble(radius);
                toServer.flush();

                // Dobijanje površine kruga sa servera
                double area = fromServer.readDouble();

                // Prikaz tekstualnog polja
                ta.appendText("Radius is " + radius + "\n");
                ta.appendText("Area received from the server is "
                        + area + '\n');
```

```
            } catch (IOException ex) {
                System.err.println(ex);
            }
        });

        try {
            // Kreiranje utičnice za vezu sa serverom
            Socket socket = new Socket("localhost", 8000);
            // Socket socket = new Socket("130.254.204.36", 8000);
            // Socket socket = new Socket("drake.Armstrong.edu", 8000);

            // Kreiranje ulaznog toka podataka za prijem podataka sa servera
            fromServer = new DataInputStream(socket.getInputStream());

            // Kreiranje izlaznog toka podataka za slanje podataka serveru
            toServer = new DataOutputStream(socket.getOutputStream());
        } catch (IOException ex) {
            ta.appendText(ex.toString() + '\n');
        }
    }
    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Napomena: Potrebno je startovati oba fajla desnim klikom na fajl i izborom opcije ***Run file*** .

PRIMER 2

Cilj ovog primera je kreiranje proste server-client aplikacije

Napraviti klasu TCPServer i TCPClient. TCPServer treba da radi non stop i da prima zahteve od klijenata. Kada klijent pošalje poruku server treba da je ispiše i pošalje informaciju klijentu da je poruka uspešno prikazana. Kada klijent primi potvrdu o zahtevu treba da se ugasi. Server treba da radi na portu 6789.

Klasa: TCPServer

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

public class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while (true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
            System.out.println("Waiting for message...");
            clientSentence = inFromClient.readLine();
            System.out.println("Received from client: " + clientSentence);
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }

    }
}
```

Klasa TCPClient:

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

public class TCPClient {

    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        Socket clientSocket = new Socket("localhost", 6789);
        DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        System.out.println("Enter the message: ");
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("Received from server: " + modifiedSentence);
        clientSocket.close();
    }
}
```

PRIMER 2 - OBJAŠNJENJE

Objašnjenje primera 2

Objašnjenje:

Cilj ovog programa je demonstracija komunikacije preko serverske i klijentske softverske utičnice (sockets) upotrebom ulazno-izlaznih (U/I) tokova podataka (streams). Server klasa TCPServer u sebi sadrži objekat ServerSocket čijem konstruktoru se prosleđuje port na kome

će raditi, u ovom slučaju 6789. Posle kreiranja softverske utičnice (socketa) na serveru, server za slušanje veza, tj. za primanje podataka preko veza računara u beskonačnoj petlji koristi iskaz:

Socket socket = serverSocket.accept();

Ovaj iskaz čeka dok se klijent ne poveže sa utičnicom servera. Klijent za podnošenje zahteva za uspostavljanje veze sa serverom korsti iskaz:

Socket socket = new Socket(serverName, port);
Kada server prihvati vezu, komunikacija između servera i klijenta se vodi na isti način kao kod U/I tokova podataka.
Da bi se dobio ulazni tok podataka i izlazni tok podataka, trebalo bi upotrebiti metod getInputStream() i getOutputStream() na priključnici, tj. objektu klase Socket ili ServerSocket.
.
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();

PRIMER 3

Cilj ovog primera je kreiranje proste server-client aplikacije za chat

Napraviti klasu Server i Client. Klasa Server treba da predstavlja server za chat koji se pokreće kao GUI i koji preko JTextArea prikazuje sve poruke pristigle od Client-a, kao i poruke koje su poslate sa servera. Client treba da se poveže na Server i da piše poruke tako da se na serveru ispisuju sve poruke koje su poslate, pri čemu se i na njegovoj strani ispisuju sve poruke koje je poslao, ali i koje su pristigle od servera.

Klasa Server:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;

import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class Server extends JFrame implements ActionListener {

    static ServerSocket server;
    static Socket conn;
```

```
    JPanel panel;
    JTextField NewMsg;
    JTextArea ChatHistory;
    JButton Send;
    DataInputStream dis;
    DataOutputStream dos;

    public Server() throws UnknownHostException, IOException {

        panel = new JPanel();
        NewMsg = new JTextField();
        ChatHistory = new JTextArea();
        Send = new JButton("Send");
        this.setSize(500, 500);
        this.setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        panel.setLayout(null);
        this.add(panel);
        ChatHistory.setBounds(20, 20, 450, 360);
        panel.add(ChatHistory);
        NewMsg.setBounds(20, 400, 340, 30);
        panel.add(NewMsg);
        Send.setBounds(375, 400, 95, 30);
        panel.add(Send);
        this.setTitle("Server");
        Send.addActionListener(this);
        server = new ServerSocket(2000, 1, InetAddress.getLocalHost());
        ChatHistory.setText("Waiting for Client");
        conn = server.accept();
        ChatHistory.setText(ChatHistory.getText() + '\n' + "Client Found");
        while (true) {
            try {
                DataInputStream dis = new DataInputStream(conn.getInputStream());
                String string = dis.readUTF();
                ChatHistory.setText(ChatHistory.getText() + '\n' + "Client:"
                        + string);
            } catch (Exception e1) {
                ChatHistory.setText(ChatHistory.getText() + '\n'
                        + "Message sending fail:Network Error");
                try {
                    Thread.sleep(3000);
                    System.exit(0);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
```

```
        if ((e.getSource() == Send)
&amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp; (NewMsg.getText() != "")) {
            ChatHistory.setText(ChatHistory.getText() + '\n' + "ME:"
                    + NewMsg.getText());
            try {
                DataOutputStream dos = new DataOutputStream(
                        conn.getOutputStream());
                dos.writeUTF(NewMsg.getText());
            } catch (Exception e1) {
                try {
                    Thread.sleep(3000);
                    System.exit(0);
                } catch (InterruptedException e2) {
                    // TODO Auto-generated catch block
                    e2.printStackTrace();
                }
            }
            NewMsg.setText("");
        }
    }

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

Klasa Client:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import javax.swing.JButton;
import javax.swing.JFrame;
import static javax.swing.JFrame.EXIT_ON_CLOSE;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class Client extends JFrame implements ActionListener {

    static Socket conn;
    JPanel panel;
    JTextField NewMsg;
    JTextArea ChatHistory;
    JButton Send;

    public Client() throws UnknownHostException, IOException {
```

```
        panel = new JPanel();
        NewMsg = new JTextField();
        ChatHistory = new JTextArea();
        Send = new JButton("Send");
        this.setSize(500, 500);
        this.setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        panel.setLayout(null);
        this.add(panel);
        ChatHistory.setBounds(20, 20, 450, 360);
        panel.add(ChatHistory);
        NewMsg.setBounds(20, 400, 340, 30);
        panel.add(NewMsg);
        Send.setBounds(375, 400, 95, 30);
        panel.add(Send);
        Send.addActionListener(this);
        conn = new Socket(InetAddress.getLocalHost(), 2000);

        /*
         * for remote pc use ip address of server with function
         * InetAddress.getByName("Provide ip here")
         * ChatHistory.setText("Connected to Server");
         */
        ChatHistory.setText("Connected to Server");
        this.setTitle("Client");
        while (true) {
            try {
                DataInputStream dis = new DataInputStream(conn.getInputStream());
                String string = dis.readUTF();
                ChatHistory.setText(ChatHistory.getText() + '\n' + "Server:"
                        + string);
            } catch (Exception e1) {
                ChatHistory.setText(ChatHistory.getText() + '\n'
                        + "Message sending fail:Network Error");
                try {
                    Thread.sleep(3000);
                    System.exit(0);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        if ((e.getSource() == Send)
&amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp; (NewMsg.getText() != "")) {

            ChatHistory.setText(ChatHistory.getText() + '\n' + "Me:"
                    + NewMsg.getText());
```

```
            try {
                DataOutputStream dos = new DataOutputStream(
                        conn.getOutputStream());
                dos.writeUTF(NewMsg.getText());
            } catch (Exception e1) {
                ChatHistory.setText(ChatHistory.getText() + '\n'
                        + "Message sending fail:Network Error");
                try {
                    Thread.sleep(3000);
                    System.exit(0);
                } catch (InterruptedException e2) {
                    // TODO Auto-generated catch block
                    e2.printStackTrace();
                }
            }
            NewMsg.setText("");
        }
    }

    public static void main(String[] args) throws UnknownHostException,
            IOException {
        Client chatForm = new Client();
    }
}
```

PRIMER 3 - OBJAŠNJENJE

Objašnjenje primera 3

Objašnjenje:

Veza i komunikacija između servera i klijenta se ostvaruje na identičan način kao u prethodnom primeru, osim što za sam prikaz i slanje poruka koristimo GUI kako bismo simulirali jednostavnu chat aplikaciju. Kao što je u objašnjenju prethodnog zadatka navedeno, komunikacija preko serverske i klijentske softverske utičnice (sockets) se odvija upotrebom ulazno-izlaznih (U/I) tokova podataka (streams).

Možete koristi DataInputStream, DataOutputStream, BufferedReader i PrintWriter da unosite ili iznosite podatke, kao što su int, double, ili String. Takođe možete koristiti tokove podataka InputStream i OutputStream koji unose i iznose bajtove.

PRIMER 4 - PRIMENA VIŠENITNOSTI U RADU SERVERA SA VIŠE KLIJENATA

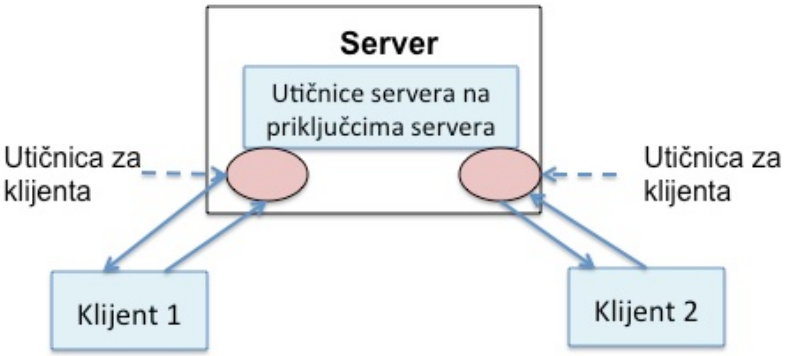
Jedan serverski program može da opslužuje više klijenata. Za vezu sa svakim klijentom se koristi po jedna posebna nit.

Vrlo često je za jedan server povezano više klijenata. Najčešće, serverski program se izvršava u jednom (serverskom) ralčunaru, a klijenti su na posebnim računarima povezanih preko

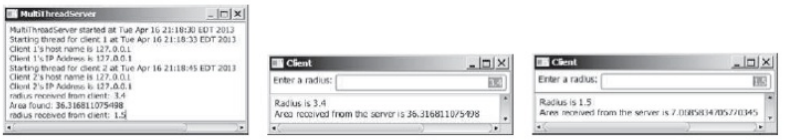
Interneta sa serverom. Za simultani rad sa više klijenata, treba da kreirate po jednu nit za svakog klijenta. To se radi na sledeći način:

```
while (true) {
   Socket socket = serverSocket.accept(); // Veza sa klijentom
   Thread thread = new ThreadClass(socket);
   thread.start();
 }
```

Slika 1 prikazuje jedan jednostavan primer povezivanja dva klijenta za jedan server, čiji listing klase MultiThreadServerje prikazan u sledećoj sekciji. Za svaki klijent je formirana posebna nit, koja prima od klijenta veličinu poluprečnika, a šalje mu nazad veličinu njegove površine kruga. Program klijenta je dat u objektu učenje “Primer klijent-server aplikacije”, ali se daje i u sledećoj sekciji ponovo.. Na slici 2 dat je prikaz prozora koji se pojavljuju na monitoru. Po jedan za svakog od dva klijenta, i jedan za server.



Slika 2.2.4 Višenitnost omogućava istovremeni rad servera sa više klijenata



Slika 2.2.5 Prikaz rada servera i dva klijenta na monitoru računara

PRIMER 4 - LISTING KLASE MULTITHREADSERVER

Svaka nit kreira ulazni izlazni tok podataka da bi primila i slala podatke klijentu

Server kreira utičnicu (socket) na utičnici (port) 8000 (linija 29) i čeka na vezi (liija 35). Kada se uspostavi veza sa klijentom, server kreira novu nit (thread) za realizaciju uspostavljene veze (linija 54). Onda čeka u beskonačnoj ***while*** petlji (linija 33-55). Niti, koje rade nezavisno jedna od druge, komuniciraju sa svojim klijentima. Svaka nit kreira ulazni izlazni tok podataka (data streams) da bi primila i slala podatke klijentu.

Ovde se ponovo navodi listink klase **Client**:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class Client extends Application {

    // UI tokovi
    DataOutputStream toServer = null;
    DataInputStream fromServer = null;

    @Override // Redefinisanje metoda start() u klasi Application
    public void start(Stage primaryStage) {
        // Panel p za natpis i  tekstualno polje
        BorderPane paneForTextField = new BorderPane();
        paneForTextField.setPadding(new Insets(5, 5, 5, 5));
        paneForTextField.setStyle("-fx-border-color: green");
        paneForTextField.setLeft(new Label("Enter a radius: "));

        TextField tf = new TextField();
        tf.setAlignment(Pos.BOTTOM_RIGHT);
        paneForTextField.setCenter(tf);

        BorderPane mainPane = new BorderPane();
        // Tekstualno polje za prikaz sadržaja
        TextArea ta = new TextArea();
        mainPane.setCenter(new ScrollPane(ta));
        mainPane.setTop(paneForTextField);

        // Kreiranje scene i njeno postavljanje na pozornicu
        Scene scene = new Scene(mainPane, 450, 200);
        primaryStage.setTitle("Client"); // Unos naziva pozornice
        primaryStage.setScene(scene); // Postavljanje scene na pozornicu
        primaryStage.show(); // Prikaz pozornice

        tf.setOnAction(e -> {
            try {
                // Dobijanje poluprečnika iz tekstualnog polja
                double radius = Double.parseDouble(tf.getText().trim());

                // Slanje poluprečnika serveru
                toServer.writeDouble(radius);
                toServer.flush();
```

```
                // Dobijanje površine kruga sa servera
                double area = fromServer.readDouble();
                // Prikaz tekstualnog polja
                ta.appendText("Radius is " + radius + "\n");
                ta.appendText("Area received from the server is "
                        + area + '\n');
            } catch (IOException ex) {
                System.err.println(ex);
            }
        });

        try {
            // Kreiranje utičnice za vezu sa serverom
            Socket socket = new Socket("localhost", 8000);
            // Socket socket = new Socket("130.254.204.36", 8000);
            // Socket socket = new Socket("drake.Armstrong.edu", 8000);

            // Kreiranje ulaznog toka podataka za prijem podataka sa servera
            fromServer = new DataInputStream(socket.getInputStream());

            // Kreiranje izlaznog toka podataka za slanje podataka serveru
            toServer = new DataOutputStream(socket.getOutputStream());
        } catch (IOException ex) {
            ta.appendText(ex.toString() + '\n');
        }
    }

     /**
     *
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Listing klase **MultiThreadServer**:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.stage.Stage;
```

```
public class MultiThreadServer extends Application {

    // Prostor za tekst za prikaz sadržaja
    private TextArea ta = new TextArea();

    // Broj klijenata
    private int clientNo = 0;

    @Override // Redefinasnje metoda start() klase Application.
    public void start(Stage primaryStage) {
        // Kreiranje scene i njeno postavljanje na pozornicu.
        Scene scene = new Scene(new ScrollPane(ta), 450, 200);
        primaryStage.setTitle("MultiThreadServer"); // Unos naslova pozornice
        primaryStage.setScene(scene); // Postavljanje scene na pozornicu
        primaryStage.show(); // Prikaz pozornice

        new Thread(() -> {
            try {
                // Kreiranje utičnice servera
                ServerSocket serverSocket = new ServerSocket(8000);
                ta.appendText("MultiThreadServer started at "
                        + new Date() + '\n');

                while (true) {
                    // Osluškivanje zahteva za novom vezom
                    Socket socket = serverSocket.accept();

                    // Povećanje broja klijenta
                    clientNo++;

                    Platform.runLater(() -> {
                        // Prikaz broja klijenta
                        ta.appendText("Starting thread for client " + clientNo
                                + " at " + new Date() + '\n');

                        // Pronalaženje naziva i IP adrese računara klijenta
                        InetAddress inetAddress = socket.getInetAddress();
                        ta.appendText("Client " + clientNo + "'s host name is "
                                + inetAddress.getHostName() + "\n");
                        ta.appendText("Client " + clientNo + "'s IP Address is "
                                + inetAddress.getHostAddress() + "\n");
                    });

                    // Kreiranje i startovanje nove niti za vezu sa klijentom
                    new Thread(new HandleAClient(socket)).start();
                }
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }).start();
    }

    // Definisanje klase niti za rad sa novom vezom.
```

```
    class HandleAClient implements Runnable {

        private Socket socket; // Povezana utičnica

        /**
         * Konstruisanje niti
         */
        public HandleAClient(Socket socket) {
            this.socket = socket;
        }

        /**
         * Izvršavanje niti
         */
        public void run() {
            try {
                // Kreiranje ulaznog i izlatnog toka podataka
                DataInputStream inputFromClient = new DataInputStream(
                        socket.getInputStream());
                DataOutputStream outputToClient = new DataOutputStream(
                        socket.getOutputStream());

                // Kontinualno služenje klijenta
                while (true) {
                    // Dobijanje poluprečnika od strane klijenta
                    double radius = inputFromClient.readDouble();

                    // Proračun površine
                    double area = radius * radius * Math.PI;

                    // Vraća nazad klijentu površinu kruga
                    outputToClient.writeDouble(area);

                    Platform.runLater(() -> {
                        ta.appendText("radius received from client: "
                                + radius + '\n');
                        ta.appendText("Area found: " + area + '\n');
                    });
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Napomena: Da biste uspeli da testirate MultiThreadServer, potrebno je pokrenuti i server I nekoliko puta klijenta desnim klikom na fajl, pa ***Run file***

PRIMER 5 - OBJEKAT KOJI SE ŠALJE

Neophodno je da program može da šalje i prima objekte poslate iz drugog programa.

Slanje i primaje objekta realizuje se primenom klasaObjectOutputStream i ObjectInputStream primenjenih na tokovima utičnica. Da bi se objekti mogli da šalju, oni moraju da bu serializovani, tj. da primenjuju interfejs ***Serializable.***

Sledeći primer pokazuje kako se mogu slati i primati objekti. Primer čine tri klase: **StudentAddress**, **StudentClient,** i **StudentServer**, čiji listinzi se daju ovde i u sledećim sekcijama. Klijentski program prikuplja informaciju o studentu sa klijenta i šalje ih serveru, kao što je pokazano na slici 1.

Slika 2.2.6 Klijent šalje serveru informaciju o studentu u objektu

Klasa **StudentAddress** sadrži informaciju o studentu, koju čine:ime, uliaca, grad, država i poštanski broj. Klasa primenjuje interejs ***Serializable.***.

Objekat **StudentAddress** može da bude poslat i primljen primenom izlaznih i ulaznih tokova objekata.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Jovana
 */
public class StudentAddress implements java.io.Serializable {

    private String name;
    private String street;
    private String city;
```

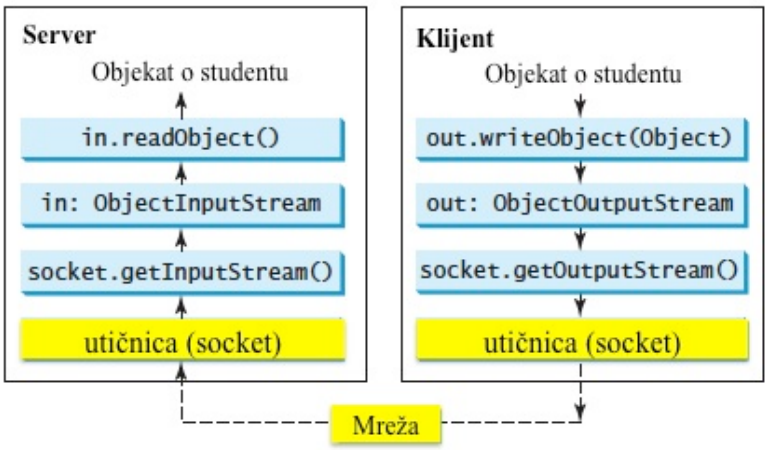
```
    private String state;
    private String zip;

    public StudentAddress() {
    }

    public StudentAddress(String name, String street, String city, String state,
String zip) {
        this.name = name;
        this.street = street;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getZip() {
        return zip;
    }

    public void setZip(String zip) {
        this.zip = zip;
```

```
    }

    @Override
    public String toString() {
        return "StudentAddress{" + "name=" + name + ", street=" + street + ",
city=" + city + ", state=" + state + ", zip=" + zip + '}';
    }

}
```

PRIMER 5 - PROGRAM KLIJENTA

Klijent koristi metod writeObject() klase ObjectOutputStream da bi poslao serveru podatke o studentu, a server ovo prima primenom metoda readObject() klase ObjectInputStream.

Klijent šalje objekat **StudentAddress** preko ***ObjectOutputStream*** objekta primenjenog na utičnici (socket) sa izlaznim tokom., a server ga prime preko objekta ObjectInputStream na utičnici sa ulaznim tokom, kao što je pokazano na slici 2. Klijet koristi metod ***writeObject()*** klase ***ObjectOutputStream*** da bi poslao serveru podatke o studentu, a server prima ovu informaciju o studentu primenom metoda ***readObject()*** klase ***ObjectInputStream.*** Listinzi klijentskog i serverskog programa su dati ovde i u sledećoj sekciji.



Slika 2.2.7 Klijent šalje serveru objekt sa informaciom o studentu

Slanje i primaje objekta realizuje se primenom klasaObjectOutputStream i ObjectInputStream primenjenih na tokovima utičnica. Da bi se objekti mogli da šalju, oni moraju da bu serializovani, tj. da primenjuju interfejs ***Serializable.***

Sledeći primer pokazuje kako se mogu slati i primati objekti. Primer čine tri klase: **StudentAddress**, **StudentClient,** i **StudentServer**, čiji listinzi se daju ovde i u sledećim sekcijama. Klijentski program prikuplja informaciju o studentu sa klijenta i šalje ih serveru, kao što je pokazano na slici 1.

Slika 2.2.8 Klijent šalje serveru informaciju o studentu u objektu

Klasa **StudentAddress** sadrži informaciju o studentu, koju čine:ime, uliaca, grad, država i poštanski broj. Klasa primenjuje interejs ***Serializable.***.

Listning klase **StudentClient:**

```
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class StudentClient extends Application {

    private TextField tfName = new TextField();
    private TextField tfStreet = new TextField();
    private TextField tfCity = new TextField();
    private TextField tfState = new TextField();
    private TextField tfZip = new TextField();

    // Dugme za slanje serveru  objekta o studenu
    private Button btRegister = new Button("Register to the Server");

    // Naziv računara ili IP adresa
    String host = "localhost";

    @Override // Redefinisanje metoda start() klase Application
    public void start(Stage primaryStage) {
        GridPane pane = new GridPane();
        pane.add(new Label("Name"), 0, 0);
        pane.add(tfName, 1, 0);
```

```
        pane.add(new Label("Street"), 0, 1);
        pane.add(tfStreet, 1, 1);
        pane.add(new Label("City"), 0, 2);

        HBox hBox = new HBox(2);
        pane.add(hBox, 1, 2);
        hBox.getChildren().addAll(tfCity, new Label("State"), tfState,
                new Label("Zip"), tfZip);
        pane.add(btRegister, 1, 3);
        GridPane.setHalignment(btRegister, HPos.RIGHT);

        pane.setAlignment(Pos.CENTER);
        tfName.setPrefColumnCount(15);
        tfStreet.setPrefColumnCount(15);
        tfCity.setPrefColumnCount(10);
        tfState.setPrefColumnCount(2);
        tfZip.setPrefColumnCount(3);

        btRegister.setOnAction(new ButtonListener());

        // Kreiranje scene  i njeno postavljanje na pozornicu
        Scene scene = new Scene(pane, 450, 200);
        primaryStage.setTitle("StudentClient"); // Unos naslova pozornice
        primaryStage.setScene(scene); // Postavljanje scene na pozornicu
        primaryStage.show(); // Prikaz pozornice
    }

    /**
     * Obrada akcije dugmeta
     */
    private class ButtonListener implements EventHandler<ActionEvent> {

        @Override
        public void handle(ActionEvent e) {
            try {
                // Uspostavljanje veze sa serverom
                Socket socket = new Socket(host, 8000);

                // Kreiranje izlaznog toka na serveru
                ObjectOutputStream toServer = new
ObjectOutputStream(socket.getOutputStream());

                // Uzimanje polja sa tekstom
                String name = tfName.getText().trim();
                String street = tfStreet.getText().trim();
                String city = tfCity.getText().trim();
                String state = tfState.getText().trim();
                String zip = tfZip.getText().trim();

                // Kreiranje objekta Student o njegovo slanje serveru
                StudentAddress s
                        = new StudentAddress(name, street, city, state, zip);
                toServer.writeObject(s);
```

```
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

PRIMER 5 - PROGRAM SERVERA

Server uz pomoć metoda readObject() prima objekat StudentAddress preko ulaznog toka objekata i zapisuje ga u datoteku.

Na strani klijenta, kad korisnik klikne dugme Regiser ili Server, klijentski program kreira utičnicu (socket) da bi se povezao sa računarom (linija 67), kreira***ObjectOutputStream***na izlaznom toku na utičnici (linije 70 i 71) i pozova metod ***writeObject()*** da bi poslao objekat **StudentAddress** na server preko izlatnog toka podataka (linija 83).

Na strani servera, kada se klijent poveže sa serverom, server kreira **ObjectInput Stream** objekat za rad sa ulaznim tokom objekata na utičnici (linije 27-28), poziva ***readObject()*** da prima objekat **StudentAddress** preko ulaznog tok aobjekata (linija 31) i i upisuje objekat u datoteku (linija 34).

Listing klase **StudentServer:**

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;


public class StudentServer {

    private ObjectOutputStream outputToFile;
    private ObjectInputStream inputFromClient;

    public static void main(String[] args) {
        new StudentServer();
    }

    public StudentServer() {
```

```
        try {
            // Kreiranje serverske utičnice (socket)
            ServerSocket serverSocket = new ServerSocket(8000);
            System.out.println("Server started ");

            // Kreiranje izlaznog toka objekata
            outputToFile = new ObjectOutputStream(
                    new FileOutputStream("student.dat", true));

            while (true) {
                // Osluškivanje zahteva za novom vezom
                Socket socket = serverSocket.accept();

                // Kreiranje ulaznog toka objekata iz utićnice (socket)
                inputFromClient
                        = new ObjectInputStream(socket.getInputStream());

                // Čitanje sa ulaza
                Object object = inputFromClient.readObject();

                // Upisivanje u detoteku
                outputToFile.writeObject(object);
                System.out.println("A new student object is stored");
            }
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            try {
                inputFromClient.close();
                outputToFile.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }


}
```

VIDEO: UDP PROGRAMIRANJE UTIČNICA

UDP Socket Programming in Java Tutorial (20,26)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

2.2 Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD STUDENTA

Cilj zadatka je provežbavanje naučenog

Zadatak 2:

Napraviti chat sistem u Javi koji će prepoznavati osnovne emotikone. Napomena: Koristiti TCP soket u Javi.

Zadatak 3:

Napraviti server koji koristi TCP soket i od klijenta prima dva broja. Brojeve treba sabrati, a rezultat vratiti klijentu.

Zadatak 4:

Napraviti server koji koristi TCP soket i od klijenta prima dva broja. Brojeve treba pomnožiti i od njih oduzeti broj 2, a rezultat vratiti klijentu.

Zadatak 5:
Napraviti TCP server koji od klijenta prima cifru, a potom cifru množi sa trenutnim kursom sa sajta nbs.rs i vraća klijentu cifru pomnoženu sa srednjim kursom.

Poglavlje 3

Klasa InetAddress

PRIMER 6: KORIŠĆENJE KLASE INETADDRESS

Serverski program upotrebljava klasu InetAddress za dobijanje informacije o IP adresi i nazivu računara klijenta.

Ako želite da saznate ko je sve povezan sa serverom, možete da koristite klasu InetAddress da bi dobili IP adresu i naziv računara klijenta. Možete da koristite sledeći iskaz u serverskom programu da bi dobili primerak klase InetAddress na utičnici (socket) koja povezuje klijenta sa serverom. .

```
InetAddress inetAddress = socket.getInetAddress();
```

IP adresu i naziv računara ožete da prikažete primenom sledećeg iskaza:

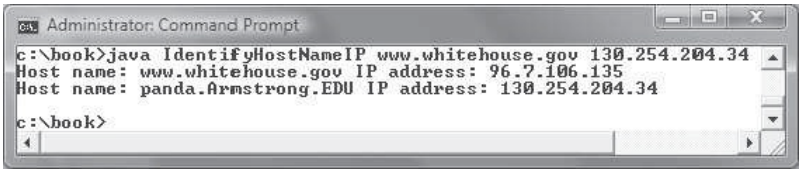
```
System.out.println("Client's host name is " +
  inetAddress.getHostName());
 System.out.println("Client's IP Address is " +
  inetAddress.getHostAddress());
```

Možete da kreirate primerak ***InetAddres*** klase sa nazivom računara i IP adrese primenom statičkog metoda ***getByName().*** Na primer, sledeći iskaz kreira primerak InetAddress klase za računar **liang.armstrong.edu.**

```
InetAddress address = InetAddress.getByName("liang.armstrong.edu");
```

Ovde se prikazuje listing klase ***IdentifyHostNameIP*** koji daje naziv računara i IP adresa na osnovu unetih argumenata preko komandne linije.

Linija 7 kreira objekat ***InteAddress*** klase dobijen pozivom metoda ***getByName().*** Pozivom metoda ***getHostName()*** i ***getHostAddress()*** (linije 8 i 9), dobijau se naziv računara i IP adresa klijenta. Na slici 1 se vidi dobijen prikaz na monitoru računata:



Slika 3.1.1 Program daje naziv računara i IP adresu klijenta

```
1 import java.net.*;
  2
  3 public class IdentifyHostNameIP {
  4   public static void main(String[] args) {
  5  for (int i = 0; i < args.length; i++) {
  6    try {
  7      InetAddress address = InetAddress.getByName(args[i]);
  8      System.out.print("Host name: " + address.getHostName() + " ");
  9      System.out.println("IP address: " + address.getHostAddress());
 10    }
 11    catch (UnknownHostException ex) {
 12      System.err.println("Unknown host or IP address " + args[i]);
 13    }
 14  }
 15   }
 16 }
```

3.1 Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD STUDENTA

Cilj je provežbavanje naučenog

Zadatak 6:

Korišćenjem klase InetAddress pronaći IP adresu sajta *metropolitan.ac.rs* i

Poglavlje 4

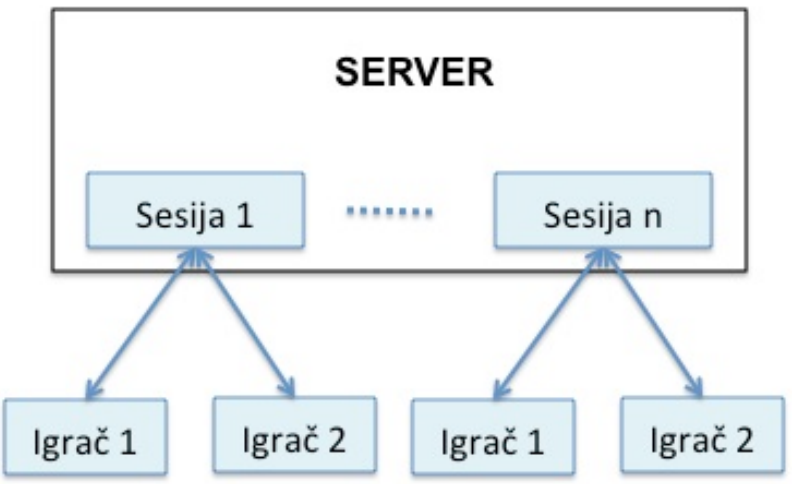
Studija slučaja: Distribuirane igre Tic-Tac-Toe

POSTAVKA PROBLEMA

Distribuirana igra Tic-Tae-Toe koristi više niti, mrežu i utičnice (sockets) i omogućava da dva igrača koriste različite računare postavljene bilo gde na Internetu.

Ovde će se pokazati kako se može napraviti program za poznatu igru Tic-Tac-Toe, ali koja je distribuisana, jer koristi više niti, mrežu i utičnice (sockets) i omogućava da dva igrača koriste različite računare postavljene bilo gde na Internetu.

Potrebno je da razvijete server sa više klijenata. Server kreira utičnicu servera i prihvata veze sa bilo koja dva igrača pri kreiranju sesije. Svaka sesija je jedna nit koja komunicira sa dva igrača i određuje status igre. Server može da postavi bilo koji broj sesija, kao što je pokazano na slici 1. Za svaku sesiju, oynačava se kao igrač br. 1 onaj koji se prvi j poveže sa serverom i sa žetonom X, a drugi koji se poveže je oynačen kao igrač 2 sa žetonom O. Server obaveštava igrače o njihovim žetonima. Kada su oba klijenta povezana, server startuje nit da bi olakšao igru između dva igrača, ponavljajući potrebne korake igre, kao što je pokazano na slici 2.



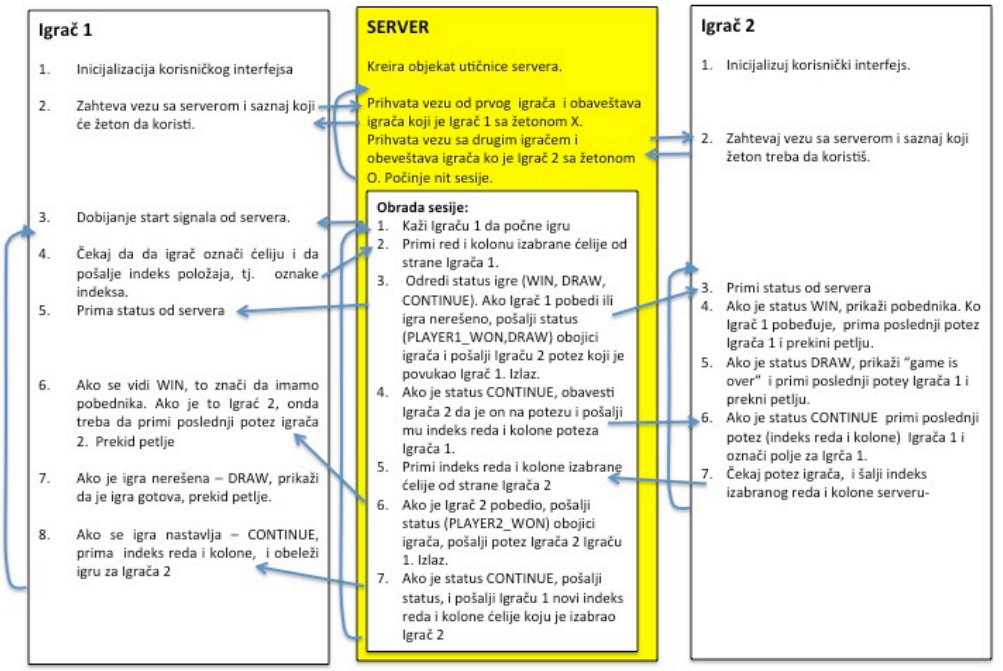
Slika 4.1 Za svaku igru server kreira po jednu sesiju sa dva igrača

SCENARIO IGRE

Server kreira nit za obradu sesije igre kada se dva igrača povežu sa serverom.

Kada su oba klijenta povezana, server startuje nit da bi olakšao igru između dva igrača, ponavljajući potrebne korake igre, kao što je pokazano na slici 2.

Server ne mora da dude grafička komponenta, ali ako ima GUI sa informaciom o igri, je lepo za korisnika. Možete staviti i prostor teksa sa klizačem za prikaz teksta u GUI. Server kreira nit za obradu sesije igre kada se dva igrača povežu sa serverom. Klijentski program je odgovoran za interakciju sa igračima. Kreira korisnički interfejs sa 9 ćelija i prikazuje nslov igre i status igre igraća u natpisima. Klasa klijenta je vrlo slična klasi za slučaj igre na jednom računaru. Ovde klijent ne određuje status igre, već samo prenosi poteze serveru i prima status igre od servera.



Slika 4.2 Scenario jedne igre, u kojoj server startuje nit koja realizuje komunikaciju sa igračima

POTREBNE KLASE

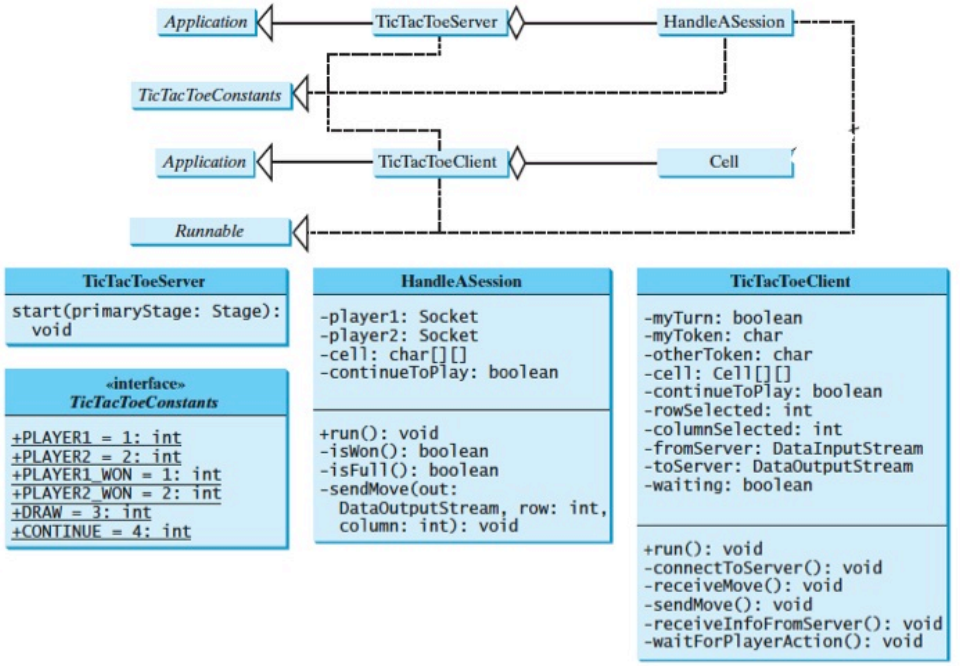
TicTaeToeServer kreira objekat klase HandleSession za svaku sesiju sa dva igrača. TicTaeToeClient kreira 9 ćelija u korisničkom interfejsu.

Treba kreirati sledeće klase:

- **TicTacToeServer** opslužuje sve klijente
- **HandleASession** olakšava igru dva igrača. Ova klasa je definisana u klasi TicTaeToeServer.java datoteci.
- **TicTacToeClient** modeluje jednog igrača
- **Cell** modeluje ćeliju igre. To je unutrašnja klasa u klasi **TicTaeToeClient.**
- **TicTacToeConstants** je interfejs koji definiše konstante koje dele sve klase

.Na slici 3 prikazan je UML dijagram sa ovim klasama, a koji pokazuje njihovu povezanost.

Listinzi ovih klasa su dati u narednim sekcijama.



Slika 4.3 TicTaeToeServer kreira objekat klase HandleSession za svaku sesiju sa dva igrača. TicTaeToeClient kreira 9 ćelija u korisničkom interfejsu.

LISTINZI PROGRAMA SERVERA I KLIJENTA

Klasa TicTaeToeServer definiše program servera, a klasa TicTaeToeClijent definiše program klijenta.

Listing klase **TicTaeToeServer:**

```
1 import java.io.*;
  2 import java.net.*;
  3 import java.util.Date;
  4 import javafx.application.Application;
  5 import javafx.application.Platform;
  6 import javafx.scene.Scene;
  7 import javafx.scene.control.ScrollPane;
  8 import javafx.scene.control.TextArea;
  9 import javafx.stage.Stage;
 10
 11 public class TicTacToeServer extends Application
 12   implements TicTacToeConstants {
 13   private int sessionNo = 1; // Broj sesije
 14
 15   @Override // Redefinisanje metoda start() klase Application
 16   public void start(Stage primaryStage) {
 17  TextArea taLog = new TextArea();
 18
 19  // Kreiranje scene i njeon postavljanje na pozornicu
 20  Scene scene = new Scene(new ScrollPane(taLog), 450, 200);
 21  primaryStage.setTitle("TicTacToeServer"); // Set the stage title
```

```
22  primaryStage.setScene(scene); // Place the scene in the stage
23  primaryStage.show(); // Display the stage
24
25  new Thread( () -> {
26    try {
27      // Kreiranje utičnice servera
28      ServerSocket serverSocket = new ServerSocket(8000);
29      Platform.runLater(() -> taLog.appendText(new Date() +
30        ": Server started at socket 8000\n"));
31
32      // Sprema da kreira sesiju ya svaka dva igrača
33      while (true) {
34        Platform.runLater(() -> taLog.appendText(new Date() +
35          ": Wait for players to join session " + sessionNo + '\n'));
36
37        // Veza sa igračem 1
38        Socket player1 = serverSocket.accept();
39
40        Platform.runLater(() -> {
41          taLog.appendText(new Date() + ": Player 1 joined session "
42            + sessionNo + '\n');
43          taLog.appendText("Player 1's IP address" +
44            player1.getInetAddress().getHostAddress() + '\n');
45        });
46
47        // Obavesti da je igrač označen kao PLAYER1
48        new DataOutputStream(
49          player1.getOutputStream()).writeInt(PLAYER1);
50
51        // Veza sa igračem 2
52        Socket player2 = serverSocket.accept();
53
54        Platform.runLater(() -> {
55          taLog.appendText(new Date() +
56            ": Player 2 joined session " + sessionNo + '\n');
57        taLog.appendText("Player 2's IP address" +
58          player2.getInetAddress().getHostAddress() + '\n');
59        });
60
61        // Obavesti igrača da je označen kap PLAYER2
62        new DataOutputStream(
63          player2.getOutputStream()).writeInt(PLAYER2);
64
65        // Prikaži ovu sesiju i inkrementalno povećaj broj sesije
66        Platform.runLater(() ->
67          taLog.appendText(new Date() +
68            ": Start a thread for session " + sessionNo++ + '\n'));
69
70        // Lansiraj novu nit za ovu sesiju dva igrača
71        new Thread(new HandleASession(player1, player2)).start();
72      }
73    }
74    catch(IOException ex) {
```

```
75      ex.printStackTrace();
76    }
77  }).start();
78   }
79
80   // Definiš klasu niti za rad sa novom sesijom dva igrača
81   class HandleASession implements Runnable, TicTacToeConstants {
82  private Socket player1;
83  private Socket player2;
84
85  // Kreiraj i inicijaliziraj ćelije
86  private char[][] cell = new char[3][3];
87
88  private DataInputStream fromPlayer1;
89  private DataOutputStream toPlayer1;
90  private DataInputStream fromPlayer2;
91  private DataOutputStream toPlayer2;
92
93  // Nastavi d  igraš
94  private boolean continueToPlay = true;
95
96  /** Konstruisanje niti  */
97  public HandleASession(Socket player1, Socket player2) {
98    this.player1 = player1;
99    this.player2 = player2;
100
101       // Initialize cells
102       for (int i = 0; i < 3; i++)
103       for (int j = 0; j < 3; j++)
104       cell[i][j] = ' ';
105     }
106
107     /** Primeni metod run() za ovu nit */
108     public void run() {
109       try {
110         // Kreiraj ulazne i izlazne tokove podataka
111         DataInputStream fromPlayer1 = new DataInputStream(
112           player1.getInputStream());
113         DataOutputStream toPlayer1 = new DataOutputStream(
114           player1.getOutputStream());
115         DataInputStream fromPlayer2 = new DataInputStream(
116           player2.getInputStream());
117         DataOutputStream toPlayer2 = new DataOutputStream(
118           player2.getOutputStream());
119
120         // Napiši nešto da obavestiš igrača 1 da počne igru
121         // Ovo samo obaveštava igrača 1 da zna da počinje
122         toPlayer1.writeInt(1);
123
124         // Kontinualno služi igrače i pripremi  izvršta j
125         // o statusu igre igračima
126         while (true) {
127           // Prijem poteza igrača 1
```

```
128           int row = fromPlayer1.readInt();
129           int column = fromPlayer1.readInt();
130           cell[row][column] = 'X';
131
132           // Proveri da li je igrač 1 pobedio
133           if (isWon('X')) {
134             toPlayer1.writeInt(PLAYER1_WON);
135             toPlayer2.writeInt(PLAYER1_WON);
136             sendMove(toPlayer2, row, column);
137             break; // Prkid petlje
138           }
139           else if (isFull()) { // Proveri da li su popunjene sve ćelije
140             toPlayer1.writeInt(DRAW);
141             toPlayer2.writeInt(DRAW);
142             sendMove(toPlayer2, row, column);
143             break;
144           }
145           else {
146             // Obavesti igrača 2 da je na potezu
147             toPlayer2.writeInt(CONTINUE);
148
149              // Pošalji red i kolonu igrača 1 igraču 2
150             sendMove(toPlayer2, row, column);
151           }
152
153           // Prijem poteza Igrača 2 R
154           row = fromPlayer2.readInt();
155           column = fromPlayer2.readInt();
156           cell[row][column] = 'O';
157
158           // Provera da li je Igrač 2 pobedio
159           if (isWon('O')) {
160           toPlayer1.writeInt(PLAYER2_WON);
161           toPlayer2.writeInt(PLAYER2_WON);
162           sendMove(toPlayer1, row, column);
163           break;
164         }
165         else {
166           // Obavesti igrača 1 da je na potezu
167           toPlayer1.writeInt(CONTINUE);
168
169           // Pošalji red i kolonu igrača 2 igraču 1
170           sendMove(toPlayer1, row, column);
171         }
172       }
173     }
174     catch(IOException ex) {
175       ex.printStackTrace();
176     }
177   }
178
179   /** Pošalji potez drugom igraču */
180   private void sendMove(DataOutputStream out, int row, int column)
```

```
    throws IOException {
      out.writeInt(row); // Šalji indeks reda
      out.writeInt(column); // Šalji indeks kolone
    }

    /** Odredi da li su sve ćelije zauzete */
    private boolean isFull() {
      for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
      if (cell[i][j] == ' ')
        return false; // Najmanje jedna ćelija nije popunjena

      // Sve ćelije su popunjene
      return true;
    }

    /** Određivanje da li je igrač sa specificiranim žetonom pobedio */
    private boolean isWon(char token) {
      // Provera svih redova
      for (int i = 0; i < 3; i++)
        if ((cell[i][0] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;am
p;amp;amp;amp; (cell[i][1] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;am
p;amp;amp;amp; (cell[i][2] == token)) {
            return true;
        }

        /** Provera svih kolona */
        for (int j = 0; j < 3; j++)
          // Pošalji red i kolonu igrača 1 igraču 2
          sendMove(toPlayer2, row, column);
        }

        // Prijem poteza Igrača 2
        row = fromPlayer2.readInt();
        column = fromPlayer2.readInt();
        cell[row][column] = 'O';

        // Prvera da li je Igrač 2 pobedio
        if (isWon('O')) {
          toPlayer1.writeInt(PLAYER2_WON);
          toPlayer2.writeInt(PLAYER2_WON);
          sendMove(toPlayer1, row, column);
          break;
        }
        else {
          // Obavesti igrača 1 da je na potezu
          toPlayer1.writeInt(CONTINUE);

          // Pošačji red i kolonu igrača 2 igraču 1
```

```
          sendMove(toPlayer1, row, column);
        }
      }
    }
    catch(IOException ex) {
      ex.printStackTrace();
    }
  }

  /** Slanje poteza drugom igraču  */
  private void sendMove(DataOutputStream out, int row, int column)
    throws IOException {
      out.writeInt(row); // Pošalji indeks red
      out.writeInt(column); // Pošalji indeks kolone
    }

    /** Određivanje da li su sve ćelije zauzete */
    private boolean isFull() {
    for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
    if (cell[i][j] == ' ')
      return false; //  najmanje jedna želija nije popunjena

    // Sve ćelije su popunjene žetonima
    return true;
  }

  /** Određivanje da li je igrać sa određenim žetonom pobedio  */
  private boolean isWon(char token) {
  // Proveri sve redove
  for (int i = 0; i < 3; i++)
    if ((cell[i][0] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[i][1] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[i][2] == token)) {
         return true;
    }

    /** Proveri sve kolone */
    for (int j = 0; j < 3; j++)
      if ((cell[0][j] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[1][j] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[2][j] == token)) {
          return true;
      }
```

```
        /** Proveri glavnu dijagonalu */
        if ((cell[0][0] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[1][1] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[2][2] == token)) {
            return true;
        }

        /** Proveri pod-dijagonalu */
        if ((cell[0][2] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[1][1] == token)
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; (cell[2][0] == token)) {
            return true;
        }

        /** Sve provereno, ali nema pobednika  */
        return false;
      }
    }
  }
```

Listing klase **TicTaeToeClient_**

```
import java.io.*;
import java.net.*;
import java.util.Date;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.scene.shape.Line;
import javafx.stage.Stage;

public class TicTacToeClient extends Application
 implements TicTacToeConstants {
  // Označava da li je igrač na potezu
  private boolean myTurn = false;

```

```
23   // Označava žeton igrača
24   private char myToken = ' ';
25
26   // Označava žeton drugog igrača
27   private char otherToken = ' ';
28
29   // Kreira i inicijalizira ćelije
30   private Cell[][] cell = new Cell[3][3];
31
32   // Kreira i inicijalizira naslov natpisa
33   private Label lblTitle = new Label();
34
35   // Kreira i incijalizira status natpisa
36   private Label lblStatus = new Label();
37
38   // Označava izabrani red i kolonu tekućeg poteza
39   private int rowSelected;
40   private int columnSelected;
41
42   // Ulazni i izlazni tokovi sa servera, odn. ka serveru
43   private DataInputStream fromServer;
44   private DataOutputStream toServer;
45
46   // Nastavak igre?
47   private boolean continueToPlay = true;
48
49   // Čeka da igrač da ozbači ćeliju
50   private boolean waiting = true;
51
52   // Naziva računara ili IP adresa
53   private String host = "localhost";
54
55   @Override // Redefinisanje metoda start() u klasi Application
56   public void start(Stage primaryStage) {
57  // Okno koje sadrži ćeliju
58  GridPane pane = new GridPane();
59  for (int i = 0; i < 3; i++)
60  for (int j = 0; j < 3; j++)
61  pane.add(cell[i][j] = new Cell(i, j), j, i);
62
63  BorderPane borderPane = new BorderPane();
64  borderPane.setTop(lblTitle);
65  borderPane.setCenter(pane);
66  borderPane.setBottom(lblStatus);
67
68  // Kreiranje scene i njeno postavljanje na pozornicu
69  Scene scene = new Scene(borderPane, 320, 350);
70  primaryStage.setTitle("TicTacToeClient"); // Unos naslova pozornice
71  primaryStage.setScene(scene); // Postavljanje scene na pozornici
72  primaryStage.show(); // Prikaz pozornice
73
74  // Veza sa serverom
75  connectToServer();
```

```
76   }
77
78   private void connectToServer() {
79  try {
80    // Kreiranje utičnice za vezu ka serveru
81    Socket socket = new Socket(host, 8000);
82
83    // Kreiranje ulaznog toka za prijem podataka sa servera
84    fromServer = new DataInputStream(socket.getInputStream());
85
86    // Kreiranje ozlaznog toka za slanje podataka ka serveru
87    toServer = new DataOutputStream(socket.getOutputStream());
88  }
89  catch (Exception ex) {
90    ex.printStackTrace();
91  }
92
93  // Kontrola igre u posebnoj niti
94  new Thread(() -> {
95    try {
96      // Dobijanje obaveštenja sa servera
97      int player = fromServer.readInt();
98
99      // Da li sa igrač 1 ili 2?
100         if (player == PLAYER1) {
101           myToken = 'X';
102           otherToken = 'O';
103           Platform.runLater(() -> {
104             lblTitle.setText("Player 1 with token 'X'");
105             lblStatus.setText("Waiting for player 2 to join");
106           });
107
108           // Prijem početnog obaveštenja sa servera
109           fromServer.readInt(); // Ignorisanje bilo kakvog unosa
110
111           // Pridružuje se drugi igrač
112           Platform.runLater(() ->
113             lblStatus.setText("Player 2 has joined. I start first"));
114
115           // Sada je moj red
116           myTurn = true;
117         }
118         else if (player == PLAYER2) {
119           myToken = 'O';
120           otherToken = 'X';
121           Platform.runLater(() -> {
122             lblTitle.setText("Player 2 with token 'O'");
123             lblStatus.setText("Waiting for player 1 to move");
124           });
125         }
126
127         // Nastavak igre
128         while (continueToPlay) {
```

```
129           if (player == PLAYER1) {
130             waitForPlayerAction(); // Ćeka potez igrača 1
131             sendMove(); // Šalje potez serveru
132             receiveInfoFromServer(); // Prijem informacije sa servera
133           }
134           else if (player == PLAYER2) {
135             receiveInfoFromServer(); // Prijem informacije sa servera
136             waitForPlayerAction(); // Ćeka potez igrača 2
137             sendMove(); // Šalje potez igrača 2 serveru
138           }
139         }
140       }
141       catch (Exception ex) {
142         ex.printStackTrace();
143       }
144     }).start();
145   }
146
147   /** Ćekanje da igrač označi želiju  */
148   private void waitForPlayerAction() throws InterruptedException {
149     while (waiting) {
150       Thread.sleep(100);
151     }
152
153     waiting = true;
154   }
155
156   /** Slanje poteza ovog igrača serveru  */
157   private void sendMove() throws IOException {
158     toServer.writeInt(rowSelected); // Slanje izabranog reda
159     toServer.writeInt(columnSelected); // Slanje izabrane kolone
160   }
161
162   /** Prima info sa servera */
163   private void receiveInfoFromServer() throws IOException {
164     // Prijem statusa igre
165     int status = fromServer.readInt();
166
167     if (status == PLAYER1_WON) {
168     // Pobeda igrača 1, yaustavljanje igre
169     continueToPlay = false;
170     if (myToken == 'X') {
171       Platform.runLater(() -> lblStatus.setText("I won! (X)"));
172     }
173     else if (myToken == 'O') {
174       Platform.runLater(() ->
175       lblStatus.setText("Player 1 (X) has won!"));
176       receiveMove();
177     }
178   }
179   else if (status == PLAYER2_WON) {
180     // Pobeda igrača 2, zaustavljanje igre
181     continueToPlay = false;
```

```
182     if (myToken == 'O') {
183       Platform.runLater(() -> lblStatus.setText("I won! (O)"));
184     }
185     else if (myToken == 'X') {
186       Platform.runLater(() ->
187       lblStatus.setText("Player 2 (O) has won!"));
188       receiveMove();
189     }
190   }
191   else if (status == DRAW) {
192     // Nema pobednika, igra je završena
193     continueToPlay = false;
194     Platform.runLater(() ->
195     lblStatus.setText("Game is over, no winner!"));
196
197     if (myToken == 'O') {
198       receiveMove();
199     }
200   }
201   else {
202     receiveMove();
203     Platform.runLater(() -> lblStatus.setText("My turn"));
204       myTurn = true; // It is my turn
205     }
206   }
207
208   private void receiveMove() throws IOException {
209     // Dobijanje poteza drugog igrača
210     int row = fromServer.readInt();
211     int column = fromServer.readInt();
212     Platform.runLater(() -> cell[row][column].setToken(otherToken));
213   }
214
215   // Unutrašnja klasa za ćeliju
216   public class Cell extends Pane {
217     // Označava red i kolonu ove ćelije table
218     private int row;
219     private int column;
220
221     // Žeton koji se koristi u ovoj ćeliji
222     private char token = ' ';
223
224     public Cell(int row, int column) {
225       this.row = row;
226       this.column = column;
227       this.setPrefSize(2000, 2000); //Šta bi se desilo bez ovoga?
228       setStyle("-fx-border-color: black"); // Postavi ivicu ćelije
229       this.setOnMouseClicked(e -> handleMouseClick());
230     }
231
232     /** Vraćanje žetona */
233     public char getToken() {
234       return token;
```

```
235     }
236
237     /** Postavljanje novog žetona */
238     public void setToken(char c) {
239       token = c;
240       repaint();
241     }
242
243     protected void repaint() {
244       if (token == 'X') {
245         Line line1 = new Line(10, 10,
246             this.getWidth() - 10, this.getHeight() - 10);
247         line1.endXProperty().bind(this.widthProperty().subtract(10));
248         line1.endYProperty().bind(this.heightProperty().subtract(10));
249         Line line2 = new Line(10, this.getHeight() - 10,
250           this.getWidth() - 10, 10);
251         line2.startYProperty().bind(
252           this.heightProperty().subtract(10));
253         line2.endXProperty().bind(this.widthProperty().subtract(10));
254
255         // Dodavanje linija u okviru
256         this.getChildren().addAll(line1, line2);
257       }
258       else if (token == 'O') {
259         Ellipse ellipse = new Ellipse(this.getWidth() / 2,
260             this.getHeight() / 2, this.getWidth() / 2 - 10,
261             this.getHeight() / 2 - 10);
262         ellipse.centerXProperty().bind(
263           this.widthProperty().divide(2));
264         ellipse.centerYProperty().bind(
265           this.heightProperty().divide(2));
266         ellipse.radiusXProperty().bind(
267           this.widthProperty().divide(2).subtract(10));
268         ellipse.radiusYProperty().bind(
269           this.heightProperty().divide(2).subtract(10));
270         ellipse.setStroke(Color.BLACK);
271         ellipse.setFill(Color.WHITE);
272
273         getChildren().add(ellipse); // Dodavanje elipse u okvir
274       }
275     }
276
277     /* Obrada događaja klika miša  */
278     private void handleMouseClick() {
279       // Ako ćelija nije zauzeta sa igračem i ako je igrač na potezu
280       if (token == ' '
&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;&amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp;amp; myTurn) {
281         setToken(myToken); // Postavlja žeton igrača u ćeliju
282         myTurn = false;
283         rowSelected = row;
284         columnSelected = column;
285         lblStatus.setText("Waiting for the other player to move");
```

```
286         waiting = false; // Upravo završen uspešan potez
287       }
288     }
289   }
290 }
```

OBJAŠNJENJE RADA PROGRAMA

Server može da simultano (istovremeno) opslužuje bilo koji broj sesija. Svaka sesija se brine o igri dva igrača.

Klasa **TicTaeToeConstants**:

```
1 public interface TicTacToeConstants {
 2   public static int PLAYER1 = 1; // Označava igrača 1
 3   public static int PLAYER2 = 2; // Označava igrača 2
 4   public static int PLAYER1_WON = 1; // Označava pobedu igrača 1
 5   public static int PLAYER2_WON = 2; // Označava pobedu igrača 2
 6   public static int DRAW = 3; // Oynačavanje poteza
 7   public static int CONTINUE = 4; // Označavanje nastavka
 8 }
```

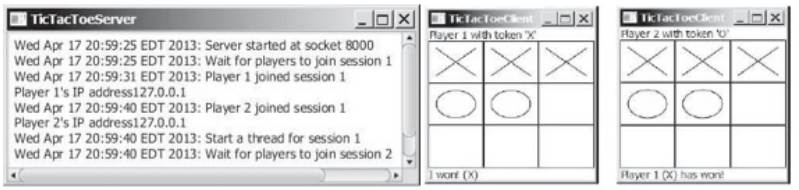
Server može da simultano (istovremeno) opslužuje bilo koji broj sesija. Svaka sesija se brine o igri dva igrača. Klijent se može primeniti i kao Java aplet. Da bi se izvršio klijent kao Java aplet sa veb prikazivača, server se mora izvršavati sa veb servera. Slika 4 prikazuje primer izvršenja programa servera i klijenta.

Interfejs **TicTacToeConstants** definiše konstante koje koriste sve klase ovog projekta. Svaka klasa koja upotrebljava ove konstante mora da upotrebljava ova interfejs. Centralno definisanje konstanti u interfejsu je opšta praksa u Javi.

Po uspostavljenju sesije, server naizmenično prima poteze od igrača. Po prijemu poteza od igrača, server određuje status igre. Ako igra nije završena, server šalje status CONTINUE, i šalje potez igrača drugom igraču. Ako ima pobednika ili ako je igra nerežena, server šalje status (PLAYER1_WON, PLAYER2_WON, ILI DROW).

Prmena Javinih mrežnih aplikacija na nivou utičnica je tačno sinhroniyovana. Operacija zahteva slanje podataka sa jedne mašine na drugu. Kao što je pokazano u ovom primeru , server i klijenti si preciyno sinhronizovani u slanju ili primanju podataka.

The implementation of Java network programs at the socket level is tightly synchronized.



Slika 4.4 a) TicTaeToeServer prihvata zahteve za uspostavljane veze i kreira sesije za opsluživanje dva igrača. b) TicTacToeClient može se izvršavati kao aplet ili kao posebna aplikacija.

Poglavlje 5

Primeri:JSoup, JavaMail biblioteka i Process klasa

PRIMER 7 - PREUZIMANJE NASLOVA VESTI

Cilj ovog primera je prikaz korišćenja JSOUP biblioteke

Napraviti program koji preuzima naslove kao i linkove do vesti sa sajta: http://www.b92.net/sport/

Napomena: koristiti Jsoup biblioteku

Jsoup biblioteku možete preuzeti sa sajta:

https://jsoup.org

Neophodno je u projekat uključiti jar fajl preuzet sa prethodno navedenog linka.

Klasa Main:

```
import java.io.IOException;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class Main {

    public static void main(String[] args) {
        try {
            Document doc = Jsoup.connect("http://www.b92.net/sport/").get();
            Elements newsHeadlines = doc.select("#tabs-1 > ul > li > a");
            for (Element e : newsHeadlines) {
                System.out.println("http://b92.net" + e.attr("href"));
                System.out.println(e.html());
            }

        } catch (IOException e) {
            System.out.println(e);
        }
    }

}
```

PRIMER 7 - OBJAŠNJENJE

Objašnjenje primera 7

Objašnjenje:

Jsoup je Java biblioteka koja služi za rad sa HTML stranama. Obezbeđuje pogodne metode za izvlačenje i manipulisanje podacima sa web strana. Sve što je potrebno da bi vesti bile preuzete sa sajta je pre svega otvoriti konekciju na samu adresu:

Document doc = Jsoup.connect("http://www.b92.net/sport/").get();

Za dobijanje elemenata koji predstavljaju naslove vesti na samom sajtu, neophodno je u hijerarhiji html tag-ova pronaći odgovarajuće i na osnovu njih izvući sve elemente:

Elements newsHeadlines = doc.select("#tabs-1 > ul > li > a");
Tag-ovi selekcije ukazuju na to da se pomenuti naslovi vesti nalaze u okviru tag-a sa id-jem tabs-1 u listi (ul > li > a, a u okviru HTML-a služi za oznaku linkova).

PRIMER 8 – KONVERZIJA KURSA PO KURSU PREUZETOG SA SAJTA NARODNE BANKE SRBIJE

Cilj primera je prikaz korišćenja JSOUP biblioteke

Napraviti program koji preuzima trenutni srednji kurs Narodne banke Srbije sa sajta: http://www.nbs.rs/static/nbs_site/gen/cirilica/30/kurs/IndikativniKurs.htm

Na osnovu pročitanog podatka prikazati za 20 evra koliko je to dinara. Napomena: koristiti Jsoup biblioteku

Jsoup biblioteku možete preuzeti sa sajta:

https://jsoup.org

Neophodno je u projekat uključiti jar fajl preuzet sa prethodno navedenog linka.

Klasa Main:

```
import java.io.IOException;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;

public class Main {

    public static void main(String[] args) {
        try {
            Document doc = Jsoup.connect("http://www.nbs.rs/static/nbs_site/gen/
cirilica/30/kurs/IndikativniKurs.htm").get();
            Elements kurs = doc.select("body > table > tbody > tr:nth-child(3) >
```

```
td:nth-child(1)");
            double kursBroj = Double.parseDouble(kurs.html().replace(",", "."));

            System.out.println("20 EURA PO DANASNJEM KURSU JE: " + kursBroj * 20);

        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

PRIMER 8 - OBJAŠNJENJE

Objašnjenje primera 8

Objašnjenje:

Za dobijanje elemenata koji predstavljaju srednji kurs Narodne banke Srbije, neophodno je u hijerarhiji html tag-ova pronaći odgovarajuće i na osnovu njih izvući sve odgovarajuće elemente:

Elements kurs = doc.select("body > table > tbody > tr:nth-child(3) > td:nth-child(1)");

Selekcija označava da se podatak koji želimo da preuzmemo nalazi na samoj strani (body) u okviru tela tabele(table > tbody) u trećem redu (tr:nth-child(3)) i prvoj ćeliji reda (td:nth-child(1)).
Dobijeni podatak konvertujemo na double, menjajući zarez tačkom i računamo koliko je dinara 20 evra po današnjem srednjem kursu.
double kursBroj = Double.parseDouble(kurs.html().replace(",", "."));
System.out.println("20 EURA PO DANASNJEM KURSU JE: " + kursBroj * 20);

PRIMER 9 – IZVUĆI SVE PODATKE O MREŽI NA RAČUNARU

Cilj primera je prikaz korišćenja Process klase

Napraviti program koji izvlači sve podatke o mreži na računaru koristeći javinu Process klasu.

Klasa IPConfig:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class IPConfig {

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

```
        new IPConfig();
    }

    public IPConfig() throws IOException {
        BufferedReader in = null;
        try {
            Process p = Runtime.getRuntime().exec("ipconfig");
            InputStream s = p.getInputStream();

            in = new BufferedReader(new InputStreamReader(s));
            String temp;

            while ((temp = in.readLine()) != null) {
                System.out.println(temp);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (in != null) {
                in.close();
            }
        }
    }
}
```

PRIMER 9 - OBJAŠNJENJE

Objašnjenje primera 9

Objašnjenje:

Cilj ovog zadatka je izvlačenje podataka o mreži na računaru koristeći se Process klasom. Objekat Process kreiramo koristeći Runtime.getRuntime().exec(“komandaprocesakojizelimodakreiramo”);

Komanda za dobijanje informacija o mreži je *ipconfig* , te za proces koji kreiramo koristimo sledeći iskaz:

Process p = Runtime.getRuntime().exec("ipconfig");

Iz toka podataka koji dobijamo iz procesa p:

InputStream s = p.getInputStream();
koristeći BufferedReader čitamo i prikazujemo podatke o mreži.

PRIMER 10 – SLANJE MAILOVA

Cilj ovog primera je kreiranje realnog servera

Prepraviti prvi zadatak tako da kada klijent šalje poruku, server tu istu poruku pošalje na Vaš email preko Gmail-a.

Napomena: neophodno je koristiti Java Mail bibloteku.

Klasa SendMail:

```
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendMail {

    final static String username = "vasuser@gmail.com";
    final static String password = "vasasifra";
    final static String to = "vasemail@gmail.com";

    public static void sendMail(String mail) {
        Properties props = new Properties();

        props.put(
                "mail.smtp.auth", "true");
        props.put(
                "mail.smtp.starttls.enable", "true");
        props.put(
                "mail.smtp.host", "smtp.gmail.com");
        props.put(
                "mail.smtp.port", "587");

        Session session = Session.getInstance(props,
                new javax.mail.Authenticator() {
                    protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(username, password);
                    }
                });

        try {

            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(username));
            message.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(to));
            message.setSubject("Testing Subject");
            message.setText(mail);

            Transport.send(message);

            System.out.println("Done");
```

```
        } catch (MessagingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Klasa TCPServer:

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPServer {

    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while (true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            System.out.println("Received: " + clientSentence);
            SendMail.sendMail(clientSentence);
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Klasa TCPClient:

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;

public class TCPClient {

    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        Socket clientSocket = new Socket("localhost", 6789);
        DataOutputStream outToServer = new
```

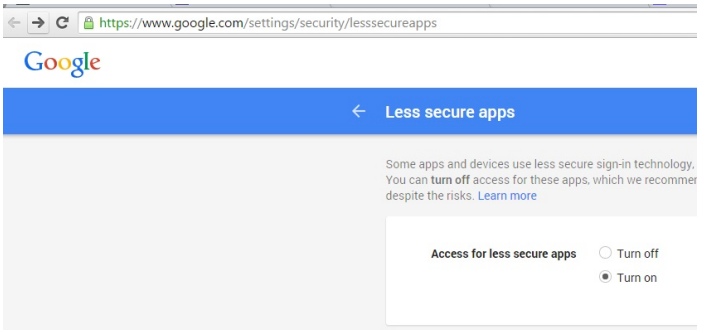
```
DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}
```

PRIMER 10 - OBJAŠNJENJE

Objašnjenje primera 10

Objašnjenje:

Ovaj zadatak ima za cilj demonstriranje korišćenja Java Mail biblioteke za slanje email-ova, simulirajući realan server. Ono što je neophodno uraditi kako bi Gmail mogao da šalje mejlove je da se izvrši podešavanje naloga sa kog se šalje tako što se uključi mogućnost slanja sa drugih aplikacija:



Slika 5.1.1 Podešavanje GMAIL

Više o biblioteci i metodama koje se koriste možete pronaći na:

http://www.oracle.com/technetwork/java/javamail/index.html

5.1 Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD STUDENTA

Cilj zadatka je provežbavanje naučenog

Zadatak 7:

Napraviti server koji će konektovanjem klijenta automatski slati email sa linkom ka Vašem CV-u okačenom na Google Drive ili DropBox

Zadatak 8:

Koristeći Jsoup preuzeti najnovije vesti sa sajta: http://www.novosti.rs/

Zadatak 9:

Preuzeti celu kursnu listu sa sajta kursna-lista.com koristeći Jsoup biblioteku.

Poglavlje 6

Domaći zadatak

DOMAĆI ZADACI

Za ove zadatke se ne daje rešenje i očekuje se da svaki student pokuša samostalno rešavanje istih

Zadatak 1

Napraviti server koji koristi TCP soket i od klijenta prima ceo broj. Server konvertuje broj u binarni, a rezultat (string) vraća klijentu.

Zadatak 2

Napraviti aplikaciju koja ucitava cene automobila sa pocetne strane sajta http://www.polovniautomobili.com, cene cuva u file i prikazuje progressbar koji se update-uje kako se cene skidaju. Koristiti posebnu nit za skidanje i update.

Zaključak

REZIME

Pouke lekcije

1. Java podržava utičnice tokova i datagrama.. Utičnice tokova upotrebljavaju TCP (Transmission Control Protocol) za prenos podataka, dok utičnice datagrama upotrebljavaju UDP (User DatagramProtocol). Kako TCP može da utvrdi izgubljene prenose podataka i da ih ponovo uradi, ti prenosi se ne mogu izgubiti i pouzdani su. Suprotno od toga, UDP ne može da garantuje gubitak prenosa.
2. Da bi kreirali server, morate prvo da dobijete utičnicu servera, upotrebom iskaza new ServerSocket(). Posle kreiranja utičnice servera, server može da počne osluškivanja veza, upotrebom metoda accept() na utičnici servera. Klijent zahteva vezu sa serverom upotrebom iskaza new Socket(serverName, port) da bi kreirao utičnicu klijenta.
3. Komunikacija preko utičnica tokova je vrlo slična komunikaciji ulaznih /izlaznih tokova polse uspostavljanja veze između servera i klijenta. Možete da dobijente ulazni tok upotrebom metoda getInputStream() i izlazni toku potrebom metoda getOutputStream() na utičnici.
4. Server često mora da radi sa više klijenata istovremeno. Možete koristiti niti za simultani rad sa više klijenata tako što ćete kreirati po jednu nit za svaku vezu.

REFERENCE

Literatura korišćena u ovoj lekciji

1. Y. Daniel Liang, Introduction to Java Programming, Comprehensive Version, Chapter 31, 10th edition, Pierson – preporučeni udžbenik