Programozás III

HÁLÓZAT

HÁLÓZATKEZELÉS

A hálózatkezeléshez használatos java csomag: java. net

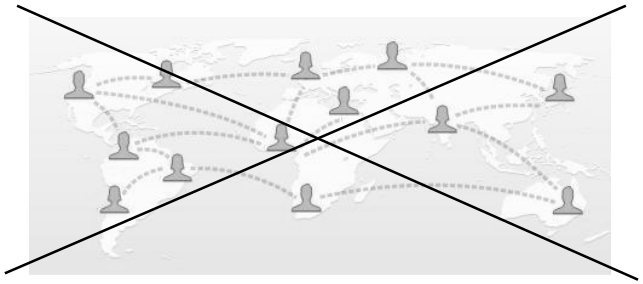
Hol találkoztunk már vele?

Pl.:

URL cim =
this.getClass().getResource("/zene/valami_zene.wav");

De pl. adott URL-ről objektumot is le tudunk tölteni.

HÁLÓZATKEZELÉS – ALAPOK



Helyette: ld. Hálózatok tantárgy vagy halozati_alapok.pdf

URL-LEL ADOTT OBJEKTUM LETÖLTÉSE (1)

```
public class URLproba {
       // Visszaadja az adott címen lévő objektumot.
   public static Object adatSzerzes(String cim){

       Object obj = "Nincs mit kiirnom.";
                // Hiba esetén az ezt a szöveget
                // tartalmazó objektumot adja vissza.
       try{
           URL url_objektum = new URL(cim);
           return obj = url_objektum.getContent();
       }
       catch (MalformedURLException e){
           System.out.println("Hibas URL cím!");
       }
       catch (IOException e){
           System.out.println("Nem sikerült a letöltés!");
                // pl. létező ftp protokoll esetén
       }
       return obj;
    }
```

```
public static void main(String[] args) {
    System.out.print("Kérem az URL címét: ");
    String url_cim = Input.readln();
    Object obj = adatSzerzes(url_cim);
        // Az objektum kiiratása
    try {
        System.out.println((String)obj);
        // csak akkor lehet String-re kényszeríteni, ha az obj
        // a "Nincs mit kiírnom" hibaüzenet, egyébként nem.
    }
    catch(ClassCastException e) {
      if (obj instanceof InputStream){
        try{
          // itt próbálja kiírni a tartalmat
          BufferedReader tartalom = new BufferedReader
                    (new InputStreamReader((InputStream)(obj)));
          boolean file_vege = false;
          while(!file_vege){
            String egy_sor = tartalom.readLine();
            if (egy_sor ==null){file_vege = true;}
            else {System.out.println(egy_sor);}
          }
        }catch (IOException ie){
          System.out.println("Hiba történt a tartalom letöltésekor");
        }
      }
      else{
        System.out.println("Nem sikerült a tartalom kiírható formára k
        System.out.println("A kapott objektum:");
        System.out.println(obj);
      }
    }   }   }
```

(2)

ld.ea_mintapeldak\URLproba.java (pl. https://pte.hu)

HÁLÓZATKEZELÉS –PROTOKOLLOK

UDP (User Datagram Protocol):
nem ad megbízható kapcsolatot egy hálózaton levő két számítógép között, mivel ez nem kapcsolat alapú, mint a TCP, inkább az adatok független csomagjait küldi az alkalmazások között. Ezeket a csomagokat adatcsomagoknak is hívjuk.

• Az adatcsomagok küldése leginkább a levelek postán át való küldéséhez hasonlít: a szállítás nem garantált, a sorrend nem fontos, és mindegyik üzenet különbözik a másiktól.

• UDP-t használó szolgáltatások:
távoli gépeken dolgozó felhasználók azonosítójának lekérdezése *(rwho)*,
fájlok hordozása *(TFTP)*,
ping.

HÁLÓZATKEZELÉS –PROTOKOLLOK

TCP (Transport Control Protocol):
az IP protokollra épülve megbízható hálózati átvitelt garantál.

• A TCP – akár egy telefontársaság – garantálja, hogy a küldött adat helyes sorrendben érkezzen meg a vevőhöz. Ha ez nem sikerül, hibajelzést küld vissza.

• Olyankor alkalmazzák, amikor a hálózaton átküldött és fogadott adatok sorrendje elengedhetetlen az alkalmazások helyes működéséhez.

• TCP-t használó legismertebb szolgáltatások:
Hyper Text Transfer Protokoll (HTTP),
fájlok átvitele távoli gépekre *(FTP: file transfer)*,
távoli gépekre történő belépés *(telnet)*,
elektronikus mail (e-mail) szolgáltatás alapját adó *SMTP* protokoll.

HÁLÓZATKEZELÉS –PROTOKOLLOK

A számítógép egy egyszerű fizikai kapcsolaton keresztül csatlakozik a hálózatra.
Minden adat ezen a kapcsolaton át érkezik, függetlenül attól, hogy az adatok a számítógép melyik programjait használják.

Honnan tudja a számítógép, hogy melyik alkalmazásnak melyik adatot kell továbbítani?

Port-ok használatával

A TCP és az UDP protokollok portokat használnak, hogy a bejövő adatokat a számítógép megfelelő programjai felé irányítsák.

HÁLÓZATKEZELÉS –PROTOKOLLOK

Az interneten küldött adat el van látva címzési információval: ez azonosítja

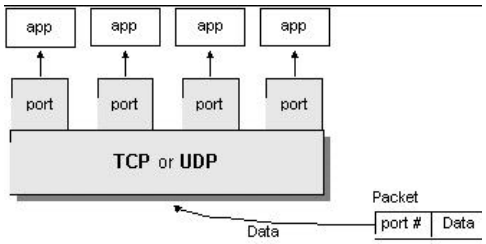
- a célszámítógépet és
- a portját.

A számítógép a 32 bites IP címmel van azonosítva, melyet arra használunk, hogy az adat a megfelelő számítógépre érkezzen meg.

A portot egy 16 bites számmal azonosítjuk, amit a TCP vagy UDP arra használ, hogy az adat a megfelelő programhoz jusson.

HÁLÓZATKEZELÉS – PORTOK

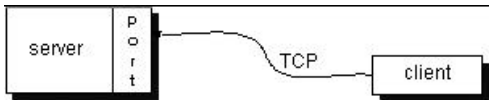
Az adatcsomag alapú kommunikációnál (UDP) az adatcsomagok tartalmazzák a célállomás portszámát, és az UDP irányítja a megfelelő helyre a csomagot.



HÁLÓZATKEZELÉS – PORTOK

A kapcsolat alapú kommunikációnál (TCP)
a szerver program leköt egy foglalatot egy jellemző port számára. Így a szerver megkap minden adatot, ami ezen a porton keresztül érkezik.

A kliens ezután a megadott porton keresztül kommunikálhat a szerverrel.



HÁLÓZATKEZELÉS – PORTOK

A portok 16 bites számként vannak ábrázolva ⇒
értékük 0 - 65535 közötti szám lehet

A 0 és 1023 közötti portok fent vannak tartva olyan ismert szolgáltatásoknak, mit például a HTTP vagy az FTP vagy más rendszerszolgáltatás. ⇒
A saját programjaink nem használhatják őket.

Egy internetes szolgáltatás elérési címe:
IP CÍM : port

Pl.: witch.pmmf.hu:2001

HÁLÓZATKEZELÉS – JAVA

A Java programok a ***java.net*** csomag osztályain keresztül használhatják a TCP vagy UDP protokollokat.

TCP-n keresztül kommunikálnak az
URL, *URLConnection*, *Socket* és *ServerSocket*
osztályok.

Az UDP protokollt használják a
DatagramPacket, *DatagramSocket* és *MulticastSocket*
osztályok.

HÁLÓZATKEZELÉS – SOCKET

Socket típusok:

- Adatfolyam socket
- Adatcsomag socket

HÁLÓZATKEZELÉS – SOCKET

A socket angol szó, jelentése:
csatlakozó, foglalat, lyuk, tok.

A socket különböző processzek (folyamatok) közötti kommunikációs eszköz, amely független attól, hogy a kommunikáló processzek azonos, vagy különböző gépen vannak-e.

A socket egy kétvégű kommunikációs hálózat egyik végpontja.

A kommunikáció formája kliens-szerver alapú.
Ez azt jelenti, hogy a szerver valamilyen jól definiált szolgáltatást nyújt, amit a kliensek igénybe vesznek.

Egy Socketnek két vége van: egy a szerver oldalán
egy a kliens oldalán.

HÁLÓZATKEZELÉS – SOCKET

Az adatfolyam socket jellemzői:

- A TCP protokollra támaszkodó, a telefonhálózathoz hasonló kapcsolat-orientált byte-stream
- Megbízható kétirányú kommunikációt tesz lehetővé
- Az adatok mindig helyes sorrendben érkeznek
- A tényleges üzenet küldéshez fel kell építeni a kapcsolatot
- Bonyolultabb felépítés $\Rightarrow$ lassabb átvitel

HÁLÓZATKEZELÉS – SOCKET

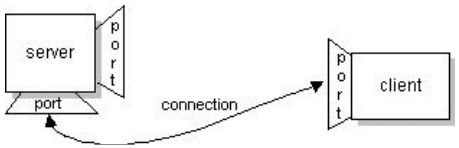
Az adatcsomag socket (datagram socket) jellemzői:

- Kapcsolat nélküli, a postai levélhez hasonló, az UDP protokollra támaszkodó megoldás
- Üzenet alapú átvitel
- Nem garantált, hogy az üzenetek megérkeznek ⇒ nem megbízható
- Egyszerűbb felépítés ⇒ gyorsabb átvitel
- Nem kell kapcsolatot felépíteni

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

Ha minden jól megy, a szerver elfogadja a kapcsolatot.

Ha a kapcsolat és a socket sikeresen létrejött, akkor a kliens használni tudja a socket-et a szerverrel való kommunikálásra.



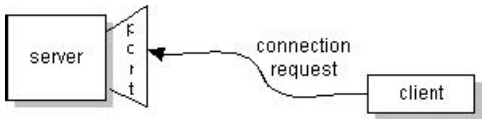
java.net.Socket : alkalmas egy kétirányú kapcsolat egyik oldalának vezérlésére
java.net.ServerSocket : figyel, és elfogadja a klienek kapcsolódását.

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

Normális esetben a **szerver** egy speciális számítógépen fut, és van egy **socket**-je, amely egy speciális **port-szám**ra van kötve.

A szerver várakozik, figyeli a socket-et, hogy érkezik-e egy klienstől kapcsolódási kérés.

A **kliens** oldalon: A kliens ismeri annak a számítógépnek a nevét, amelyiken a szerver fut, és annak a portnak a számát, amelyikre a szerver kapcsolódik.



AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

Ha a socket révén létrejött a kapcsolat a kliens és a szerver között, akkor az üzenetváltás egy adatfolyamon folyik. Adatokat olvasni az input folyamból lehet, írni pedig az output stream-be lehet.

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

A szerver lépései:

1. A szerver lefoglal egy TCP portot, amelyen a továbbiakban a kliensek rákapcsolódási kéréseit várja (egyben közzéteszi a lefoglalt TCP port azonosítóját, hogy a kliensek eltaláljanak hozzá.)

2. A szerver ezután elkezd várakozni egy kliens kapcsolat-felvételi kérelmére.

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

A kliens lépései:

1. Lefoglal egy TCP portot, amelyen keresztül majd felveszi a kapcsolatot a szerverrel.

2. Hozzákapcsolódik ahhoz a géphez, amelyen a szerver fut, azon a gépen belül pedig ahhoz a TCP porthoz, amelyen a szerver a kliensek rákapcsolódási kérelmére vár (ennek a TCP portnak az azonosítóját tette közzé a szerver az 1. lépése után).

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

A szerver lépései (folyt.):

3. Ha egy kliens közli a szerverrel kapcsolat-felvételi szándékát, akkor a szerver felveszi a kapcsolatot a klienssel, és kiszolgálja azt (a szerver feladatának megfelelő módon).

4. Miután a szerver kiszolgálta a klienst (vagy legalábbis elindította a klienst kiszolgáló folyamatot), újabb klienst fogadhat, és munkáját a második pontnál folytathatja.

Amennyiben egy kliens kiszolgálása nagyon hosszú ideig tart, akkor a szerver a 3. lépésben megteheti, hogy elindít egy új programszálat a kliens kiszolgálására, és az eredeti szerver-programszál kész egy újabb kliens fogadására.

AZ ADATFOLYAM-SOCKETEK MŰKÖDÉSE

A kliens lépései (folyt.):

3. A felépített kommunikációs vonalon adatokat fogadhat a szervertől, illetve küldhet a szervernek.

4. Lebontja a szerverrel felépített kapcsolatát (és ha a TCP portra tovább már nincs szüksége, akkor visszaadhatja azt az operációs rendszernek, hogy más folyamat is lefoglalhassa).

A kliens és a szerver kommunikációs TCP portjának az összekapcsolása után a TCP kommunikációs kapcsolat kétirányú adatátvitelre képes.

AZ ADATCSOMAG-SOCKETEK MŰKÖDÉSE

Az összeköttetés-mentes kapcsolatok megszervezése sokkal egyszerűbb:

A kapcsolat minden résztvevőjének létre kell hoznia egy UDP kommunikációs portot, ahol fogadhatja a többiektől érkező csomagokat, illetve erről a portról küldhet csomagokat a kommunikáció többi résztvevőjének.

A kommunikáció minden résztvevője egy tetszőleges azonosítóju UDP portot használhat. A kommunikáció megvalósíthatóságának fontos feltétele, hogy a kommunikáló felek ismerjék a kommunikációs partnerük címét.

Az UDP protokollra épülő kommunikációnak nehézkes a gyakorlati megvalósíthatósága,mert nincs garantálva, hogy egy ilyen módon elküldött csomag megérkezik a rendeltetési helyére.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – szerver

1. lépés

```
int server_port;
server_port = 12345;
ss = new ServerSocket(server_port);
```

Az ss a ServerSocket osztály egy példánya, amelynek egyik konstruktora azt a szerver portot várja paraméterül, amelyre a szerver rákapcsolódik és várja a kliensek kapcsolódásait.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – szerver

2. lépés

```
while (online) {
    s = ss.accept();
    kiszolgal();
}
```

A szerver egy „végtelen” hurokban keringve várja a kliensek kapcsolódási kérelmeit. A ServerSocket objektum accept() metódusa fogadja kliensek kapcsolódási kérelmeit.

Ha kapcsolódik egy kliens, akkor létrehoz egy összeköttetés alapú s kapcsolatot (socketet). A kliens és a szerver közötti üzenetváltás ezen az s socketen kereszül történik.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – szerver

3. lépés kiszolgál:

```
String keres,valasz;

BufferedReader input = new BufferedReader(
   new InputStreamReader(s.getInputStream()));
PrintWriter output = new PrintWriter(s.getOutputStream());

keres = input.readLine();
valasz = "Válasz";
output.println(valasz);
output.flush();          <-- fizikailag is kiküldjük a választ
```

Az üzenetváltásokhoz streameket(folyamokat) kell definiálni. Kétféle folyam lesz: a bejövő(input) és a kimenő(output) folyam. Bejövő folyamot a BufferedReader osztály segítségével készítünk úgy, hogy elkérjük az s socket bejövő hálózati folyamát. Kimenő folyamot a PrintWriter segítségével definiálunk úgy, hogy elkérjük az s socket kimenő folyamát.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – szerver

4. lépés

```
s.close();
ss.close();
```

A kliens és a szerver az s socketen keresztül kommunikálnak. Ha a kiszolgálás megtörtént, akkor az s kapcsolatot le kell zárni a close() metódussal (még a kiszolgáló ciklus [while] végén). Ezután a szerver tovább várja a többi kliens kapcsolódását. Amennyiben a szervert szeretnénk leállítani, úgy az ss szerver socketet is le kell zárni annak close() metódusával.

(Természetesen az input/output csatornákat is illik lezárni.)

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – kliens

3. lépé
kérele

```
String keres,valasz;

BufferedReader input = new BufferedReader(
       new InputStreamReader(s.getInputStream()));
PrintWriter output = new PrintWriter(s.getOutputStream());

output.println(keres);
output.flush();

valasz = input.readLine();
```

Az s socketen keresztül pontosan ugyanúgy működik az üzenet-váltás, mint a szervernél, amikor már kapcsolódott a kliens, csak most értelemszerűen a kérést kell kiírni a kimenő adatfolyamba, és a szerver válaszát kell olvasni a bejövő adatfolyamból.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – kliens

1-2. lépés

```
String ip_cim;
int port;
Socket s;
ip_cim = "localhost";
port = 12345;
s = new Socket(ip_cim, port);
kerelem();
```

A kliensben is létre kell hoznunk egy socketet, ami hozzá-kapcsolja a klienst a szerverhez.

Ehhez 2 paraméterre lesz szükségünk: a szerver címére és a szerver számítógép azon portjára, amelyen a szerver alkalmazás várakozik a kliensek kapcsolódásaira.

JAVA MEGVALÓSÍTÁS

Összeköttetés-alapú kapcsolat (TCP) – kliens

4. lépés

```
s.close();
```

Az s socket close() metódusával lehet lezárni a hálózati kapcsolatot.

(És persze itt is lezárjuk a csatornákat.)

JAVA MEGVALÓSÍTÁS

Nem hiányzik valami?

Hálózatkezelés során rengeteg hiba adódhat ⇒

Kötelező kivételkezelés

Egy kapcsolatot mindenképpen le kell zárni, függetlenül attól, hogy kivétel adódott. ⇒ A Socket lezárását a finally ágban végezzük, de ha csak lehet, a try blokk fejében.

Ld.: szerver_vazlat.txt; kliens_vazlat.txt;

stream_kezeles_vazlat.txt

JAVA MEGVALÓSÍTÁS

Összeköttetés-mentes kapcsolat (UDP)

Önálló feldolgozás!

JAVA MEGVALÓSÍTÁS

És most már minden rendben?

A szervernek általában több klienst kell egyszerre kiszolgálnia (de persze, nem kötelező)

⬇

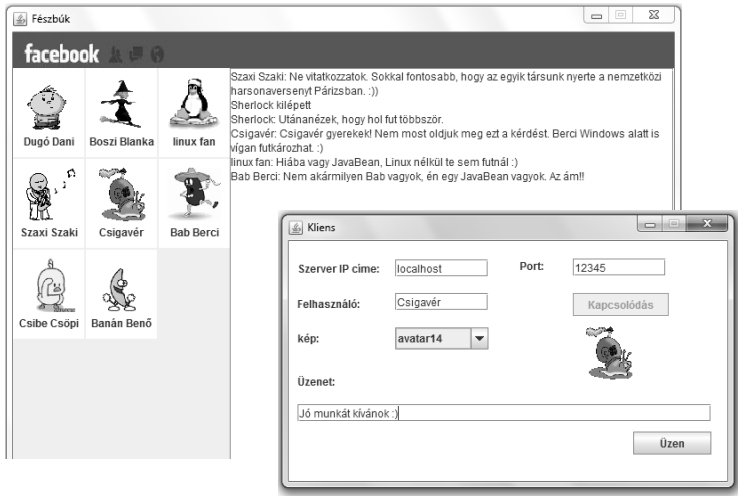
Szálkezelés kell.

Ld.: tobbszalu_szerver_vazlat.txt

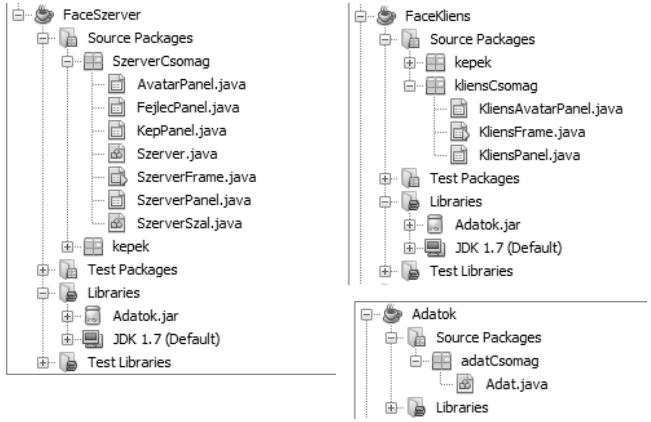
És mi a helyzet a szerverrel?

Általában azt is külön szálként definiáljuk – grafikus felület esetén feltétlenül. Miért?

PÉLDA – SAJÁT FÉSZBÚK



PÉLDA – SAJÁT FÉSZBÚK



PÉLDA – A KLIENS NÉHÁNY METÓDUSA

```
private void kapcsolGombActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        kapcsolatKialakitas();
        azonositoAdatokKuldese();
        if (socket.isConnected()) {
            kapcsolGomb.setEnabled(false);
            uzenGomb.setEnabled(true);
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "kapcsolatfelvételi hiba");
    }
}
```

```
            this.addWindowListener(new WindowAdapter() {

                @Override
                public void windowClosing(WindowEvent e) {
                    kliensPanel1.kliensLezar();
                }
            });
```

PÉLDA – SAJÁT FÉSZBÚK

A kliens lépései:

1. Kapcsolódik.
2. Elküldi az azonosításra szolgáló adatokat (név, kép).
3. Üzenget.
4. Kilép.

A szerver lépései:

1. Várakozik, majd fogadja a kliens jelentkezését.
2. Fogadja az azonosításra szolgáló adatokat, és megjeleníti őket.
3. Fogadja az üzenetet és megjeleníti.
4. Lezárja a klienssel való kapcsolatot.
5. A szerver maga is lezárul.

PÉLDA – A KLIENS NÉHÁNY METÓDUSA

```
private void kapcsolatKialakitas() throws Exception {
    String ipCim = ipMezo.getText();
    int port = Integer.valueOf(portMezo.getText());

    socket = new Socket(ipCim, port);

    if (objOut == null) {
        objOut = new ObjectOutputStream(socket.getOutputStream());
        out = new PrintWriter(socket.getOutputStream());
    }
}
```

```
   private void azonositoAdatokKuldese() throws Exception {
           nick = nevMezo.getText();
           avatar = avatarok.get(kepCombo.getSelectedIndex());

           Adat adat = new Adat(nick, avatar);
           objOut.writeObject(adat);
           objOut.flush();
   }
```

PÉLDA – A KLIENS NÉHÁNY METÓDUSA

```
private void uzenGombActionPerformed(java.awt.event.ActionEvent evt) {
    String uzenet = uzenetMezo.getText();
    if (!uzenet.isEmpty()) {
        out.println(uzenet);
        out.flush();
        uzenetMezo.setText("");
        uzenetMezo.grabFocus();
    } else {
        JOptionPane.showMessageDialog(this, "Gépeljen üzenetet");
    }
}  public void kliensLezar() {
       String uzenet = Adat.ZARSZO;
       if (socket != null) {
           out.println(uzenet);
           out.flush();
           try {
               socket.close();
               System.out.println("lezárva");
           } catch (IOException ex) {
               Logger.getLogger(KliensFrame.class.getName()).log(Lev
           }
       }
   }
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
public class SzerverFrame extends javax.swing.JFrame  {

//    private Szerver szerver = new Szerver();
//    helyette: singleton tervezési minta

    private Szerver szerver = Szerver.getPeldany();

    public SzerverFrame() {
        initComponents();
        this.setSize(750, 700);
        this.setTitle("Fészbúk");
        szerver.setKepPanel(kepPanel1);
        szerver.setSzerverPanel(szerverPanel1);
        szerver.szerverInditas();
        this.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                szerver.szerverLezaras();
                szerver = null;
                System.exit(0);
            }
        });
    }
```

PÉLDA – AZ ÁTKÜLDÖTT ADATOK

```
public class Adat implements Serializable {

    private String nick;
    private ImageIcon avatar;
    public static final String ZARSZO = "bye";

    public Adat(String nick, ImageIcon avatar) {
        this.nick = nick;
        this.avatar = avatar;
    }

    public ImageIcon getAvatar() {
        return avatar;
    }

    public String getNick() {
        return nick;
    }
}
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
public class Szerver extends Thread {

    private final int port = 12345;
    private volatile ServerSocket serverSocket;
    private volatile boolean online = true;
    private List<SzerverSzal> szalak = new CopyOnWriteArrayList<>();
    private SzerverPanel szerverPanel;
    private KepPanel kepPanel;
    private static Szerver peldany = null;

    private Szerver() {
    }

    public static Szerver getPeldany() {
        if (peldany == null) {
            peldany = new Szerver();
        }
        return peldany;
    }
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
public void szerverInditas() {
    try {
        try {
            serverSocket = new ServerSocket(port);
        } catch (Exception e) {
            System.out.println("Hiba:" + e.getMessage());
        }
        System.out.println("A szerver a " + port + " porton varja a klien

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Hibás port");
    }
    this.start();
}
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
public synchronized void szerverLezaras() {
    online = false;
    // lezárjuk a még élő szálakat
    for (SzerverSzal szal : szalak) {
        try {
            szal.lezar();
        } catch (IOException ex) {
            System.out.println("nem tudtam lezárni");
            Logger.getLogger(SzerverPanel.class.getName()).log(Level.SEVERE
        }
    }
}
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
@Override
public void run() {
    Socket socket;
    SzerverSzal szal;
    while (online) {
        try {
            if (!serverSocket.isClosed()) {
                socket = serverSocket.accept();
                szal = new SzerverSzal(socket, szerverPanel, kepPanel);
                szal.start();
                szalak.add(szal);
            }
        } catch (IOException ex) {
            Logger.getLogger(SzerverFrame.class.getName()).log(Level.SEVERE
        }
    }
}
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
class SzerverSzal extends Thread {

@Override
public void run() {

    try {
        BufferedReader in;
        PrintWriter out;
        String be, uzenet;
        ObjectInputStream oin = new ObjectInputStream(socket.getInputStream());

        Adat adat = (Adat) oin.readObject();
        nick = adat.getNick();
        kepPanel.kirak(adat);

        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream());

        while (!socket.isClosed()) {
            be = in.readLine();

            if (be.equals(Szerver.ZARSZO)) {
                in.close();
                out.close();
                socket.close();
                uzenet = nick + " kilépett";
                kepPanel.kilep(adat);

            } else {
                uzenet = nick + ": " + be;
            }

            szerverPanel.uzen(uzenet);
        }

    } catch (IOException | ClassNotFoundException ex) {
        Logger.getLogger(SzerverSzal.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

PÉLDA – A SZERVER NÉHÁNY METÓDUSA

```
Adat adat = (Adat) oin.readObject();
nick = adat.getNick();
kepPanel.kirak(adat);

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
out = new PrintWriter(socket.getOutputStream());

while (!socket.isClosed()) {
    be = in.readLine();

    if (be.equals(Szerver.ZARSZO)) {
        in.close();
        out.close();
        socket.close();
        uzenet = nick + " kilépett";
        kepPanel.kilep(adat);

    } else {
        uzenet = nick + ": " + be;
    }

    szerverPanel.uzen(uzenet);
}
```