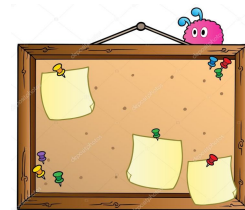
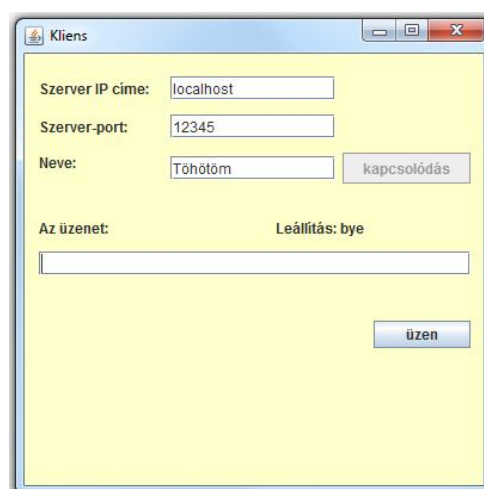
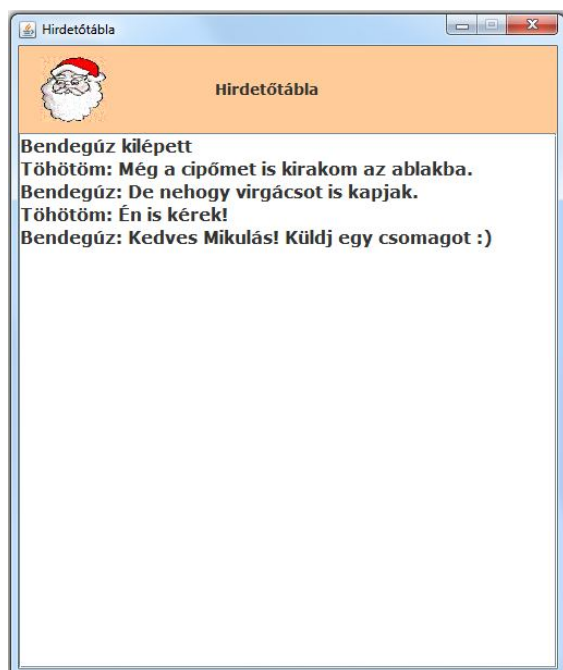


## Faliújság



A Mikulás a szerver oldalon várja a diákok jelentkezését. Amíg senki sem jelentkezik, addig kissé idegesen rángatózik a szája, de az első jelentkező után megnyugszik.

Írja meg azt a klienst, amelyről be tudunk jelentkezni hozzá, és üzenetet is tudunk hagyni az üzenő falán. És persze, azt a szervert is, amely fogadni tudja ezeket az üzeneteket.



A gombok értelemszerűen legyenek aktívak/inaktívak, az üzenetet pedig az enter lenyomásával is el lehessen küldeni.

Tudnivalók: Előbb meg kell adni a szerver IP-címét, a port most fixen 12345.

Ugyancsak adja meg a nevét.

Ezek után a kapcsolódás gomb segítségével kapcsolódjon a szerverhez.

A kapcsolódás gomb hatására jöjjön létre a szerverrel való kapcsolat, ezután a gomb legyen inaktív. A kapcsolat létrejöttékor a szerver fogadja a megadott nevet, majd visszaküldi azt a szót, amelyet megüzenve a szervernek, lezárul a kliens és a szerver kapcsolata. Ez a kapcsolat akkor is záródjon le, ha az egész alkalmazást zárjuk be. A szervertől kapott szó jelenjen is meg a kliens felületén.

Az üzen gomb hatására, illetve az Enter begépelésekor lehet üzenetet küldeni.

A kliensnek az a dolga, hogy egyetlen String-et átküldjön, a szerver majd tudja kezelni azt. (Úgy kell megírni a szervert, hogy tudja.)

Természetesen itt már két külön projektben célszerű megírni a szervert és a klienst.

## Megoldásrészletek:

### Kliens:

A megoldásban szét kell bontani a kapcsolódást az üzenetküldéstől, illetve a leállítástól. Néhány részlet:

A kapcsolódás gomb hatása:

```
private void btnKapcsolodasActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        String ipCim = txtIp.getText();  
        int port = Integer.valueOf(txtPort.getText());  
  
        // A socket, out globális, mert másik metódus is használja.  
        socket = new Socket(ipCim, port);  
        out = new PrintWriter(socket.getOutputStream(), true);  
  
        // Az in csak itt használatos, lehet lokális.  
        try (BufferedReader in =  
            new BufferedReader(new InputStreamReader(socket.getInputStream()))) {  
  
            String nev = txtNev.getText();  
            if (nev.isEmpty()) {  
                JOptionPane.showMessageDialog(this, bundle.getString("nevhiany"));  
            } else {  
                out.println(nev);  
  
                kilepes = in.readLine();  
                lblKilepes.setText(bundle.getString("felirat") + " " + kilepes);  
                btnUzen.setEnabled(true);  
                btnKapcsolodas.setEnabled(false);  
            }  
        }  
    } catch (NumberFormatException | IOException | HeadlessException ex) {  
        // lblKilepes.setText("Nem sikerült a kapcsolódás");  
        lblKilepes.setText(bundle.getString("hiba"));  
    }  
}
```

Az üzen gomb hatása:

```
private void btnUzenActionPerformed(java.awt.event.ActionEvent evt) {  
    uzen();  
}  
  
private void uzen() {  
    out.println(txtUzenet.getText());  
    if (txtUzenet.getText().equals(kilepes)) {  
        btnKapcsolodas.setEnabled(true);  
        btnUzen.setEnabled(false);  
    }  
    txtUzenet.setText("");  
}
```

Célszerű megoldani azt is, hogy az enter megnyomásának hatására is elmenjen az üzenet:

```
private void txtUzenetKeyPressed(java.awt.event.KeyEvent evt) {  
    if(evt.getKeyCode() == KeyEvent.VK_ENTER) {  
        uzen();  
    }  
}
```

Végül azt is meg kell oldani, hogy a szerver akkor is tudomást szerezzen a kilépésről, ha a kliens felületéről az ablaklezáró gomb megnyomásával akarunk távozni.

Ezért saját `windowClosing()` metódust kell írunk. Ezt a frame-n tehetjük meg:

```
public static void main(String args[]) {  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new KliensFrame().start();  
        }  
    });  
}  
  
private void start() {  
    setVisible(true);  
    this.addWindowListener(new WindowAdapter() {  
        @Override  
        public void windowClosing(WindowEvent e) {  
            kliensPanel1.lezar();  
        }  
    });  
}
```

A panel lezáró metódusa:

```
void lezar() {  
    if(socket != null) {  
        try {  
            // értesíti a szervert a kilépésről  
            out.println(kilepes);  
            out.close();  
            socket.close();  
            System.out.println("lezártam");  
        } catch (IOException ex) {  
            Logger.getLogger(KliensPanel.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
}
```

## Néhány gondolat a szerver oldal megvalósításához:

1. A szervert külön szálként kell megírni, mert különben nem tud működni a grafikus interfész, hiszen a szerver minden kapacitást lefoglalna. (Általában is legtöbbször külön szálon futtatják a szervert.)

Ennek `run()` metódusában kell megoldani azt, hogy a szerver várja a kliensek jelentkezését, és ha jelentkezik egy, akkor annak kiszolgálására létrehozzon egy `SzerverSzal` típusú példányt, ami a klienssel való kommunikációért felel – erről kicsit később bővebben lesz szó.

Ebben a `Szerver` osztályban lehet megírni a szerverindítást és a szerver lezárását. A szerver indítására vonatkozó metódusban egy adott porton létrehozza a `ServerSocket` példányt és elindítja ezt a szálát. A lezáró metódusban pedig lezárja a még aktív szerverszálakat, majd magát a szerver-socketet is.

Mindkét metódust a frame-ből hívhatjuk, az indítást annak `main()` metódusából, a lezáráshoz pedig hasonlóan felül kell definiálni a `windowClosing()` metódust, mint a kliens esetén.

Ismétlésképpen (no meg logikus is) singletonként kezeljük a szervert

2. A `SzerverSzal` osztály felel majd egy-egy klienssel való kommunikációért. Paraméterként át kell adni neki a kapcsolódáskor létrejött socket-et, illetve az üzenőfelületet biztosító panel-példányt. (Esetleg a panel helyett lehet vezérlő is, aki kellően bátor ahhoz, hogy ebben az új környezetben is próbálkozzon az MVC szemlélettel.)

Ennek `run()` metódusa valósítja meg a protokollt, azaz fogadja a kientől jövő üzeneteket a feladatban leírt módon. (Először a nevet olvassa be, visszaküldi a leállító szót, utána pedig ciklusban fogadja a küldött üzeneteket mindaddig, amíg a leállító szót nem küldi a kliens.)

3. Végül kell az a közös felület, amelyen üzengetnek egymásnak. Ezt egy panel felületére rakott szövegmezővel lehet megoldani. Az egyes szerverszálak az üzenet megkapásakor ennek a panelnek a kiíró metódusát hívják meg, és megkérik, hogy írja ki az adott üzenetet.

4. És természetesen kell a frame. ☺

Néhány kódrészlet:

```

public class Szerver extends Thread {

    private final int port = 12345;
    private volatile ServerSocket serverSocket;
    private List<SzerverSzal> szalak = new CopyOnWriteArrayList<>();
    private SzerverPanel szerverPanel;

    private static Szerver peldany = null;

    private Szerver() {
    }

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

    @Override
    public void run() {
        Socket socket;
        SzerverSzal szal;

        while (serverSocket != null && !serverSocket.isClosed()) {
            try {
                socket = serverSocket.accept();
                System.out.println("kapcsolódtak");
                szal = new SzerverSzal(socket, szerverPanel);
                // ennek hatására változik meg a Mikulás képe
                szerverPanel.kepValtas();
                szal.start();
                szalak.add(szal);
            } catch (IOException ex) {
                Logger.getLogger(SzerverFrame.class.getName()).log(Level.SEVERE,
                    null, ex);
            }
        }
    }

    public void szerverInditas() {

        try {
            serverSocket = new ServerSocket(port);
        } catch (IOException e) {
            System.out.println("Hiba:" + e.getMessage());
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Hibás port");
        }
        System.out.println("A szerver a " + port + " porton varja a klienseket");

        this.start();
    }
}

```

```

public class SzerverSzal extends Thread {

    private Socket socket;
    private static String lezar = "bye";
    private SzerverPanel szerverPanel;

    public SzerverSzal(Socket socket, SzerverPanel szerverPanel) {
        this.socket = socket;
        this.szerverPanel = szerverPanel;
    }

    @Override
    public void run() {
        BufferedReader in;
        PrintWriter out;
        String nev, be, uzenet;

        try {
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);

            nev = in.readLine();

            out.println(lezar);

            while (! socket.isClosed()) {
                be = in.readLine();
                uzenet = nev + ": " + be;
                if (be.equals(lezar)) {
                    in.close();
                    out.close();
                    socket.close();
                    uzenet = nev + " kilépett";
                }
                szerverPanel.uzen(uzenet);
            }

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

        public static void setLezar(String lezar) {
            SzerverSzal.lezar = lezar;
        }

        void lezar() throws IOException {
            socket.close();
        }
    }
}

```

A szerverindítást a frame `main()` metódusából hívott `indit()` metódusából hívjuk meg:

```

private void indit() {
    this.setVisible(true);
    szerver = Szerver.getPeldany();
    szerver.setSzerverPanel(szerverPanel1);
    szerver.szerverInditas();
}

```

A szerverpanel `uzen()` metódusa pedig kiírja a szövegmezőbe az üzenetet:

```

public void uzen(String uzenet) {
    txtHirdetoMezo.insert(uzenet+"\n", 0);
}

```

Ha ezt sikeresen megoldotta, akkor egy további feladat:



Oldja meg a lokalizációt. (Természetesen a többi üzenetet is.)

