Egy csipetnyi Swing és egy leheletnyi JFreeChart

Hogyan készítsünk mutatós felhasználói felületet pillanatok alatt Javaban? Tippek és trükkök a platformfüggetlenség jegyében.

A grafikus alkalmazások a számítógéppel most ismerkedők számára mindig barátságosabb eszközöknek bizonyulnak, mint a parancssoros programok. Éppen ezért, ha feltesszük, hogy a világegyenletet megoldó szoftverünknek nem kizárólag a legszakavatottabb kezek alatt is jó szolgálatot kell tennie, érdemes felruháznunk valamilyen ***grafikus felhasználói felülettel (GUI, graphical user interface)***. Nézzünk körül, milyen fejlesztési eszközök állnak rendelkezésünkre!

A ***Linuxvilág*** hűséges olvasói számos megoldást láttak már a problémára, mind más írók, mind jómagam tollsából. A lap hasábjain jelentek meg írások a ***Tk*** eszközkészletről, és ennek több programozói környezetben előforduló változatáról, így a ***Perl/Tk***-ról és a ***Tcl/Tk***-ról. Olvashattunk a ***Qt*** könyvtárról és a ***GTK***-ról is, melyek ***C/C++***-ból felhasználható függvénykönyvtárak. Természetesen a felsorolás a teljesség igénye nélkül készült.

Ezen eszközkészletek egy része szkriptekből felhasználható. Ezek egyik problémája, hogy nem előfordíthatók, így a futási idő ezeknél a lehető legnagyobb. Léteznek kerülőutak ennek kiküszöbölésére, de ezek a módszerek még nem kiforrottak. Az előfordított nyelvekkel az a gond, hogy még a forráskód szintű hordozhatóság biztosítása is nagy körültekintést és komoly programozói gyakorlatot igényel. Létezik ezek után megoldás?

Nem tettem volna fel a költői kérdést, ha nem igen volna a válasz. A megoldást pedig a ***Java*** programnyelv jelenti. Látom lelki szemeim előtt, ahogy most százan és ezren húzzák fel egyszerre az orrukat. Be kell látni, a Java nem egy paripa. Igen, lassabb a ***C++***-nál, és sokkal lassabb a ***C***-nél. Cserébe viszont bájtkód szinten hordozható a különböző operációs rendszerek között, és egy nagyon kényelmesen használható, gazdag függvénykönyvtárral bíró objektum-orientált környezet. Továbbá az elvakult szkripthívők is beláthatják, hogy a Java gyorsabb egy átlagos szkriptnyelvnél.

Fontos, hogy belássuk, a cél határozza meg az eszközt, és ennek mindig így kell lennie. Ha követelmény a hordozhatóság, kézenfekvő megoldást jelenthet a ***Java***. A bájtkóddá történő fordítás természetesen nem egyetemes válasz minden kérdésre, így sebességben sohasem fogja utolérni a natív kódot. Ezzel szemben a Java használatával az, hogy mely operációs rendszerek képesek futtatni alkalmazásunkat, kizárólag annak a függvénye, hogy elkészült-e már a ***Java Virtuális Gép (JVM, Java Virtual Machine)*** az adott platformra.

Nézzük tehát, mit kínál a ***Java***. Kezdetben grafikus felhasználói felületek létrehozására az ***AWT (Abstract Windowing Toolkit)*** állt rendelkezésünkre. Az eszközkészlet absztrakt jellegét az adja, hogy kizárólag olyan elemek találhatók meg benne, amelyek minden, ***JVM***-et futtató operációs rendszer grafikus felületén előfordulnak, és ezek megjelenítéséért a gazda rendszer felelős. Vagyis egy nyomógomb ***Windows*** alatt ***windowsosan***, ***Unix*** alatt „motifosan” néz ki. Ez azt is jelenti sajnos, hogy a grafikus felületek által biztosított grafikus elemek legszűkebb metszete érhető csak el a programozó számára.

Ezt követte a ***Swing*** megjelenése. A ***Swing*** egy olyan részhalmaza a ***Java*** függvénykönyvtárnak, amelyet teljesen ***Javaban*** írtak. Egyáltalán nem használja a gazda rendszer nyújtotta elemeket, ami azt jelenti, hogy mindent saját maga rajzol ki. Ebből eredően a ***Swinget*** használó Java alkalmazások minden rendszer alatt ugyanúgy néznek ki. A kinézet stílusok révén befolyásolható, ez az úgynevezett ***„Look&Feel”***. Ezzel jelen írás nem foglalkozik, alkalmazásunk az alapértelmezett ***Metal*** kinézettel bír.

Aki használta már az ***AWT***-t, könnyen megbarátkozik a ***Swinggel*** is, mivel a régi elemek új neve csak annyiban változik, hogy ott áll egy „J” betű az elején. Vagyis például `Frame` helyett `JFrame`-et fogunk használni. A névrokonság nem önkényes, nagyon sok elem régi metódusai nem változtak, és ezért ugyanúgy használhatók, mint korábban. Viszont néhány helyen alapvető változások történtek, így a `JFrame` esetében is. Nem hívható meg többek között egy `JFrame`-re az `add()` metódus egy elem hozzáadásához, mint a régi `Frame` esetében. Ezért mindig legyen nyitva egy böngészőablakban a ***Java API*** programozás közben!

Jelen írás keretében egy menüvel rendelkező grafikus alkalmazást fogunk létrehozni. A menüből modulokat lehet kiválasztani, és mindig a kiválasztott modul jelenik meg az ablakban. Ezért a főprogram mellett minden modul önálló osztályt fog képviselni. Lesz egy olyan modulunk, amely egy diagrammot rajzol ki, ráadásul dinamikusan változó adatsorral. Vágjunk is rögtön bele, mert elég sok dolgunk van!

A főprogram

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

public class Monitor implements ActionListener {

    private JMenuItem kilepes;
    private JCheckBoxMenuItem altalanos, memoria,
    ➥nevjegy;
    private Container ablakTartalom;
    private CardLayout elrendezesKezelo;

    private static final String ALTALANOS =
    ➥"Általános információk";
    private static final String MEMORIA = "Memória
    ➥használat";
    private static final String NEVJEGY = "Névjegy";

    public Monitor() {

      JFrame ablak = new JFrame("Monitor");

      JMenuBar menuSor = new JMenuBar();
      JMenu fajlMenu = new JMenu("Fájl");
      JMenu modulMenu = new JMenu("Modul");
      ButtonGroup modulCsoport = new ButtonGroup();
      altalanos = new JCheckBoxMenuItem(ALTALANOS,
      ➥true);

altalanos.setAccelerator(KeyStroke.getKeyStroke
➥("control A"));
      altalanos.addActionListener(this);
      modulCsoport.add(altalanos);
      modulMenu.add(altalanos);
      memoria = new JCheckBoxMenuItem(MEMORIA);
      memoria.setAccelerator(KeyStroke.getKeyStroke
      ➥("control M"));
      memoria.addActionListener(this);
      modulCsoport.add(memoria);
      modulMenu.add(memoria);
      nevjegy = new JCheckBoxMenuItem(NEVJEGY);
      nevjegy.setAccelerator(KeyStroke.getKeyStroke
      ➥("control N"));
      nevjegy.addActionListener(this);
      modulCsoport.add(nevjegy);
      modulMenu.add(nevjegy);
      fajlMenu.add(modulMenu);
      fajlMenu.addSeparator();
      kilepes = new JMenuItem("Kilépés");
      kilepes.setAccelerator(KeyStroke.getKeyStroke
      ➥("control K"));
      kilepes.addActionListener(this);
      fajlMenu.add(kilepes);
      menuSor.add(fajlMenu);
      ablak.setJMenuBar (menuSor);

      ablakTartalom = ablak.getContentPane();
      elrendezesKezelo = new CardLayout();
      ablakTartalom.setLayout(elrendezesKezelo);
      ablakTartalom.add(new MonitorAltalanos(),
      ➥ALTALANOS);
      ablakTartalom.add(new MonitorMemoria(),
      ➥MEMORIA);
      ablakTartalom.add(new MonitorNevjegy(),
      ➥NEVJEGY);

      ablak.setDefaultCloseOperation
      ➥(JFrame.EXIT_ON_CLOSE);
      ablak.setBounds(100, 100, 640, 480);
      ablak.setVisible(true);

    }

    public void actionPerformed(ActionEvent esemeny) {

      Object forras = esemeny.getSource();

      if (forras == altalanos) {
          elrendezesKezelo.show(ablakTartalom,
          ➥ALTALANOS);
      } else if (forras == memoria) {
          elrendezesKezelo.show(ablakTartalom,
          ➥MEMORIA);
      } else if (forras == nevjegy) {
          elrendezesKezelo.show(ablakTartalom,
          ➥NEVJEGY);
      } else if (forras == kilepes) {
          System.exit(0);
      }

    }

    public static void main(String[] args) {

      new Monitor();

    }

}
```

Bár kizárólag a ***Swing*** által biztosított eszközkészletből dolgozunk, vannak osztályok, amelyek kizárólag a jó öreg ***AWT***-ből érhetők el. Így az eseménykezeléshez használt `ActionListener` interfész, amely a `java.awt.event` csomag része, vagy a `CardLayout` nevű, elrendezéskezelőt osztály, melyet a `java.awt` csomagban találunk. Ezért importáljuk a névtérbe a `java.awt`, a `java.awt.event`, és a `javax.swing` csomagok összes elemét.
`Monitor` nevű osztályunk megvalósítja az `ActionListener` interfészt, mivel ennek lesz a felelőssége a menüben egy elemen történő kattintás eseményének a kezelése. Három publikus tagfüggvénye van: a konstruktor, amely a grafikus felületet építi fel, az `actionPerformed()`, amely az előbb említett eseményeket dolgozza fel, illetve egy statikus `main()`, melynek nincs más feladata, mint hogy létrehozzon egy példányt az alkalmazásból. Az osztály privát tagváltozóit első használatukkor, a konstruktor bemutatásánál részletezem.
A konstruktor feladata a grafikus felület felépítése. Első sorában létrehoz egy `JFrame` objektumot, ez fogja képviselni az ablakunkat. Az átadott paraméter az ablak címe.

Ezt követi egy nagyobb rész, amiben a menüt hozzuk létre. Egy igen egyszerű menüsort építünk fel, egyetlen `Fájl` menüvel, amiben van egy `Modul` lenyíló menü valamint egy `Kilépés` menüpont. A `Modul` menüből további menüpontok választhatók ki.

Egy ***Swing*** menü a következőképpen fest: egy `JMenuBar` objektum képviseli a teljes menüsort, ennek létrehozása után kell hozzárendelni az ablakhoz annak `setJMenuBar()` metódusával. A menüsor menükből áll, melyeket egy-egy `JMenu` objektum képvisel. Egy menü további menüket is tartalmazhat, ekkor ezek lenyíló menükként jelennek meg. Az egyes menüpontokat `JMenuItem` objektumok jelentik. A `JMenuItem` ősosztálya az `AbstractButton`, emiatt van `addActionListener()` metódusa, és eseménykezelése egy közönséges nyomógombhoz hasonló.

Alkalmazásunkban a Modul almenü jelölőnégyzettel ellátott menüpontokból áll, melyeket a `JCheckBoxMenuItem` osztály valósít meg. Ez a `JMenuItem` osztályból származik. A menüpontok közül mindig pontosan egy aktív, hiszen a felhasználó az ablakban mindig egy modult lát, a menü az ezek közti váltást valósítja meg. Ezt a `ButtonGroup` osztály használatával értem el. Egy `ButtonGroup` objektum egy gombcsoportot képvisel, melyhez az add() metódussal lehet hozzávenni egy nyomógombot. Az objektum felelőssége, hogy a csoportba tartozó gombok közül mindig csak az egyik legyen kiválasztva.

Nem menü a menü gyorsbillentyűk nélkül. A `JMenuItem` osztály, s így a `JCheckBoxMenuItem` osztály is, rendelkezik egy `setAccelerator()` metódussal. Ez egy `KeyStroke` objektumot vár paraméterként és a megadott billentyűkombinációt hozzárendeli ahhoz a menüponthoz, amelyre meghívták. A `KeyStroke` objektum létrehozása jelen esetben az osztály `getKeyStroke` statikus metódusának meghívásával történik, amely a paraméterében szövegesen megfogalmazott billentyűkombinációból készít egy `KeyStroke` objektumot.

Lássuk tehát sorról sorra, mi történik a menü felépítésekor! Elsőként készítünk egy menüsort, melyet `menuSor`-nak nevezünk el. Ezután létrehozunk két menüt, az egyiket `fajlMenu`-nek, a másikat `modulMenu`-nek nevezzük el. Ne felejtsük el, egy menün belül található lenyíló menü is ugyanolyan, mint az őt tartalmazó. Ezután létrehozunk egy gombcsoportot, és mivel a Modul menü elemeit fogjuk majd egybe ezzel, `modulCsoport`-nak nevezzük el.

Ezután három, nagyon hasonló lépéssorozat következik, a `Modul` almenü három menüpontjának létrehozásához. Előbb létrehozzuk a menüpontot, mint egy `JCheckBoxMenuItem` objektumot. Az első konstruktorában jelezzük, hogy már ki van választva. A menüpontokhoz beállítunk egy gyorsbillentyűt. Ezután jelezzük, hogy a menüpont kiválasztásához tartozó eseménykezelőt jelen osztály tartalmazza. Hozzávesszük a gombcsoporthoz a menüpontot. Végül hozzáadjuk a `modulMenu`-höz. Miután mindhárom `JCheckBoxMenuItem` elkészült, s ezzel feltöltöttük a `modulMenu`-t, a `fajlMenu` objektum `add()` metódusát meghívjuk a `modulMenu`-vel és így hozzávesszük a menühöz. Ezután az `addSeparator()` tagfüggvény segítségével beteszünk egy elválasztó vonalat a Modul menü alá. Végül elkészítjük a Kilépés menüpontot, amely mindössze annyiban különbözik a Modul menü elemeitől, hogy nem rendelkezik jelölőnégyzettel, és így nem is tartozik a gombcsoportba.

A menü létrehozását követő rövidebb rész az ablak tartalmát építi fel. Fontos különbség az ***AWT***-ben található `Frame`-hez képest, hogy a `JFrame`-re nem hívható meg közvetlenül az `add()` metódus. Először le kell kérdezni az ablaktartalmat képviselő `Container` objektumot és ennek lehet használni az add() tagfüggvényét. Először tehát ezt kérdezzük le. Majd létrehozunk egy `CardLayout` elrendezéskezelőt. Ennek az a különlegessége, hogy a tartalmazott objektumok közül mindig csak egyet mutat, a többit háttérben tartja. Neve is onnan ered, hogy hasonlítható egy kártyapaklihoz, melynek csak a legfelső eleme látható.

Az ablaktartalom elrendezéskezelőjét beállítjuk a létrehozott objektumra a `setLayout()` metódussal. Majd egyesével felvesszük a modulokat az `add()` metódussal, mindegyiknek azt a szöveges nevet adva, amely a menüben is szerepel. Tehát a nagybetűs állandók, az `ALTALANOS`, a `MEMORIA`, és a `NEVJEGY` nem csupán a `JCheckBoxMenuItem` létrehozásakor játszottak szerepet, mint a menüpontok feliratai, hanem a hozzájuk tartozó kártyalap nevei is egyben. Ezzel a névvel lehet később hivatkozni arra a kártyalapra, amelyet felülre kívánunk tenni.

Minden modulunk a `JPanel` osztályból ered, ezért adhattuk őket hozzá az ablaktartalomhoz ilyen egyszerűen. A konstruktor végén még néhány egyszerűbb beállítást találunk. Meghatározzuk az alapértelmezett műveletet akkor, ha az ablak bezárását kezdeményezi a felhasználó az ablakkezelőn keresztül. Beállítjuk az ablak méreteit, pozícióját, és végül láthatóvá tesszük.

Az `actionPerformed()` metódus a menüpontok valamelyikén történő kattintáskor fut le. Előbb meghatározzuk az esemény forrását, majd ettől függően elvégezzük a szükséges műveletet. Megjelenítjük a kívánt modult, vagy kilépünk a programból. Látható, hogy a menüpontok feliratával azonos nevű oldalak segítségével ez milyen egyszerűen megvalósítható.

Lássuk most az „Általános információk modult”!. Ez táblázatos formában mutatja be a rendszertulajdonságokat leíró változókat.

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

public class MonitorAltalanos extends JPanel {

    private class AdatModell extends
    ➥AbstractTableModel {

      private Vector adatok;

      public AdatModell() {
          Properties rendszerTulajdonsagok =
          ➥System.getProperties();
          adatok = new
```

```
        ➥Vector(rendszerTulajdonsagok.size());
        Enumeration kulcsok =
        ➥rendszerTulajdonsagok.keys();
        while (kulcsok.hasMoreElements()) {
      Object kulcs =
      ➥kulcsok.nextElement();
      Vector adat = new Vector(2);
      adat.add(kulcs);
      adat.add(rendszerTulajdonsagok.get(kulcs));
      adatok.add(adat);
        }
    }

    public int getRowCount() { return
    ➥adatok.size(); }

    public int getColumnCount() { return 2; }

    public Object getValueAt(int sor, int oszlop) {
        Vector sorVektor = (Vector)
        ➥adatok.get(sor);
        return sorVektor.get(oszlop);
    }

    public String getColumnName(int sor) {
        if (sor == 0) return "Kulcs";
        return "Érték";
    }

  }

  public MonitorAltalanos() {

    setLayout(new BorderLayout());
    JTable tablazat = new JTable(new
    ➥AdatModell());
    JScrollPane gorgethetoNezet = new
    ➥JScrollPane(tablazat);
    add(gorgethetoNezet, BorderLayout.CENTER);

  }

}
```

A modul szép példája a `JTable` osztály használatának. A feladat egy görgethető táblázat létrehozása. Ezt a mondatot a konstruktorban írjuk le. Beállítjuk az elrendezéskezelőt egy `BorderLayout` objektumra. Ezután létrehozunk egy táblázatot egy adatmodellel. Az adatmodellt egy belső osztály írja le. Ezt követően megalkotjuk a táblázat görgethető nézetét. Végül a panel közepére betesszük ezt az elemet.
A belső osztály az adatmodellt írja le. Ezzel elkülönül maga a `JTable` és az adatokat tartalmazó objektum. Ez azért nagyon hasznos, mert így az adatok ábrázolása teljes mértékben ránk van bízva, olyan szerkezetet használunk, amilyet csak szeretnénk. Ebben az adatmodellben egy `Vector` objektumot veszünk igénybe, amely további `Vector`-okat tartalmaz. Ezen `Vector`-ok pedig két `String` elemből, egy kulcsból és egy értékből állnak.

Az `AdatModell` konstruktora a `System` osztály `getProperties()` metódusa segítségével lekérdezi a rendszertulajdonságokat. Sajnos ez közvetlenül nem használható fel adatforrásnak, mert a benne található elemek nem rendezhetők sorba. Ezért átírjuk az összes elemét a fentebb említett `Vector`-ba, ami már egyértelmű sorrendet jelent. A lekérdezés után tehát létrehozunk egy akkora `Vector`-t, ahány eleme van a `rendszerTulajdonsagok`-nak.
Ezt követően lekérdezzük a `rendszerTulajdonsagok` kulcsait, és egy ciklusban bejárjuk őket. Minden iterációban létrehozunk egy 2 hosszú `Vector`-t, amit megtöltünk adattal, és hozzáadjuk az adatok nevű `Vector`-unkhoz.
Miután az `AdatModell` osztály kiterjeszti az `AbstractTableModel`-t, néhány metódust biztosítanunk kell. Ezek: a `getRowCount()`, ami a sorok számát adja vissza, a `getColumnCount()`, ami az oszlopok számát adja meg, illetve a `getValueAt()`, ami egy konkrét elem értékével szolgál. A `getColumnName()` nem kötelezően megvalósítandó, de ha nem definiáljuk felül, az oszlopok címei A, B, C, stb. lesznek.
Lássuk most a „Memória használat” modult! Ez egy dinamikus diagrammon mutatja be a ***Java Virtuális Gép*** memóriahasználatát.

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

import org.jfree.chart.*;
import org.jfree.data.time.*;

public class MonitorMemoria extends JPanel
➥implements ActionListener {

    private Runtime kornyezet;
    private TimeSeries osszesMemoria,
    szabadMemoria;

    public MonitorMemoria() {

      kornyezet = Runtime.getRuntime();

      osszesMemoria = new TimeSeries("Összes
      ➥Memória", Millisecond.class);
      osszesMemoria.setHistoryCount(30000);
      szabadMemoria = new TimeSeries("Szabad
      ➥Memória", Millisecond.class);
      szabadMemoria.setHistoryCount(30000);
      TimeSeriesCollection adatSor = new
      ➥TimeSeriesCollection();
      adatSor.addSeries(osszesMemoria);
      adatSor.addSeries(szabadMemoria);

      JFreeChart diagramm =
      ➥ChartFactory.createTimeSeriesChart(
          "A Java Virtuális Gép memória
          ➥használata",
          "Idő",
          "Memória",
```

```
            adatSor,
            true, true, false
        );

        ChartPanel diagrammPanel = new
        ➥ChartPanel(diagramm);

        setLayout(new BorderLayout());
        add(diagrammPanel, BorderLayout.CENTER);

        Timer utemezo = new Timer(100, this);
        utemezo.start();

    }

    public void actionPerformed(ActionEvent
    ➥esemeny) {
        szabadMemoria.add(new Millisecond(),
        ➥kornyezet.freeMemory());
        osszesMemoria.add(new Millisecond(),
        ➥kornyezet.totalMemory());
    }

}
```

Ez a modul felhasználja a `JFreeChart` *Java* csomagot, melyet a ***http://www.jfree.org/*** oldalon keresztül lehet beszerezni. Használatához mindössze a ***jcommon-x.x.x.jar*** és a ***jfreechart-x.x.x.jar*** állományok elérési útvonalát kell betenni a `CLASSPATH` környezeti változóba, vagy paranccsorban mind a fordítónak, mind a futtatókörnyezetnek a `-cp` kapcsolóval átadni. A `JFreeChart` egy nagyon gazdag környezet mindenféle grafikon egyszerű előállításához és ráadásul ingyenes is.
Tehát miután importáltuk a névtérbe az `org.jfree.chart` és az `org.jfree.data.time` csomagok elemeit, nézzük meg, hogyan is működik ez az osztály. Megvalósítja az `ActionListener` interfészt, mivel az felhasznált memóriára vonatkozó adatok periodikus lekérdezéséhez a ***Swing*** `Timer` osztályát használjuk, és ez a megadott időközönként meghívandó eljárást egy olyan osztály képében várja, amely rendelkezik az `actionPerformed()` tagfüggvénnyel.
Az osztály egy konstruktorból és egy `actionPerformed()` metódusból áll. Vannak továbbá privát tagváltozói, ezek a `kornyezet`, az `osszesMemoria`, és a `szabadMemoria`.
A konstruktor lekérdezi a környezetet, létrehozza a diagrammot, beállítja és elindítja az időzítőt.
Az `actionPerformed()`, amely minden tizedmásodpercben lefut, lekérdezi a környezettől az összes- és a szabad memóriát, és ezeket az értékeket hozzáadja az `osszesMemoria` és a `szabadMemoria` adatforrásokhoz.
A konstruktor mindenek előtt beállítja a `kornyezet` változót a `Runtime` osztály `getRuntime()` metódusa alapján. Létrehozunk két `TimeSeries` objektumot, melyek az adatforrásokat fogják jelenteni a `osszesMemoria`, illetve a `szabadMemoria` változóknak. Mindkettőnek beállítjuk, hogy a 30 másodpercnél régebbi adatokat ne vegye figyelembe. Létrehozunk továbbá egy `adatSor` változót, melyhez hozzáadjuk a két adatforrást.

Ezután létrehozzuk a diagrammot, megadva a címét, a tengelyek címeit, az adatsort, és néhány apróságot. Készítünk a diagrammból egy panelt, beállítjuk az elrendezéskezelőt, és hozzáadjuk a panelt. Ezután készítünk egy ütemezőt, amely a megadott objektum `actionPerformed()` metódusát hívja meg tizedmásodpercenként, majd elindítjuk az ütemezőt.
Végezetül a teljesség kedvéért lássuk a névjegy panelt. Noha meg vagyok győződve, hogy ezek után egyetlen sora sem jelent újdonságot.

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

public class MonitorNevjegy extends JPanel {

    public MonitorNevjegy() {

        final String sorTores =
        ➥System.getProperty("line.separator");

        setLayout(new BorderLayout());

        JTextArea szoveg = new JTextArea();
        szoveg.setEditable(false);
        szoveg.append("Monitor v0.1" + sorTores);
        szoveg.append("==========" + sorTores +
        ➥sorTores);
        szoveg.append("Java alkalmazás a Linuxvilág
        ➥olvasói számára");
        szoveg.append(sorTores + sorTores + "Fülöp
        ➥Balázs" + sorTores);
        szoveg.append("2005.");

        add(szoveg, BorderLayout.CENTER);

    }

}
```

Egy közönséges szövegdobozba írom bele a névjegy tartalmát. Ami talán érdekes lehet, az az, hogy a sortörést nem „\n"-el oldottam meg, bár a ***Java*** intelligenciája miatt úgy is működött volna bármely operációs rendszer alatt, hanem lekérdeztem a sortörést jelentő szövegfüzért a rendszertulajdonságok táblából.
Ebben az írásban elég sok kódot láthatsz, amit érdemes tanulmányozni. Ezt azért hangsúlyozom, mert kizárólag a leírásból nem fogsz rájönni mindenre. Feltétlenül töltsd le a ***Java API***-t a ***Sun*** oldaláról (***http://java.sun.com/***), ha még nem tetted meg, és próbálkozz! Ne csak azzal, amit itt látsz, hanem azzal is, amit kigondolsz! Sok örömet kívánok a ***Java***-hoz.

Fülöp Balázs (admin@guardware.com)
21 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mrožek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.