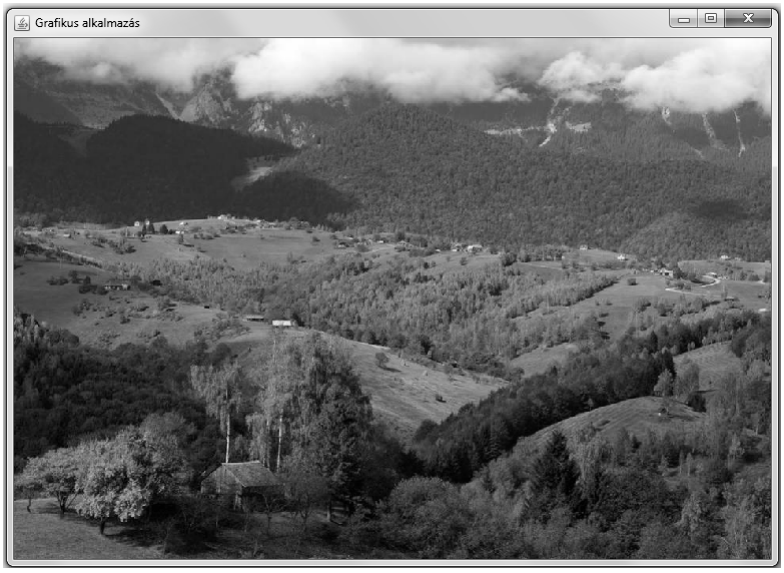
Programozás III

GAFIKA

GRAFIKA

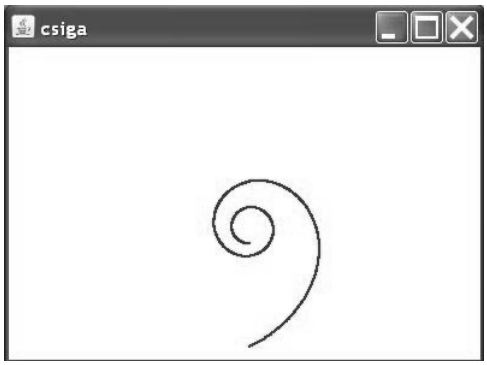


RAJZOLÁS – GRAFIKA HASZNÁLATA

Rajzolni az awt csomag Graphics osztályának metódusaival tudunk.

A **java.awt.Graphics** osztály néhány metódusa:

drawString()
drawRect()
drawOval()
fillRect()
fillOval()
…stb…



RAJZOLÁS SWING FELÜLETEN

A Java minden egyes komponenshez automatikusan felkínál egy Graphics típusú objektumot.

Elvileg rajzolhatnánk így:

```
JComponent komponens = new …
Graphics g = komponens.getGraphics();
    // Elkérjük a komponenstől a grafikus felületét
g.fillOval(…);
```

Ekkor a rajz csak egyszer jelenik meg, a komponens újrarajzolásakor eltűnik.

Sőt, törlődik! ☺

RAJZOLÁS SWING FELÜLETEN

A helyes megoldás:
Felülírjuk a JComponent osztály paintComponent(Graphics g) metódusát. A metódus a paraméterében kínálja fel a komponens grafikus objektumát – erre rajzolhatunk.

Az alkalmazás minden olyan esetben automatikusan meghívja a **paintComponent**() metódust, amikor a célfelületet frissítenie kell. (átméretezés, mozgatás, stb.) – egyébként pedig a **repaint**() metódus hatására frissül.

```
protected void paintComponent(Graphics g){
    super.paintComponent(g);
    g.fillOval(…); }
```

RAJZOLÁS SWING FELÜLETEN

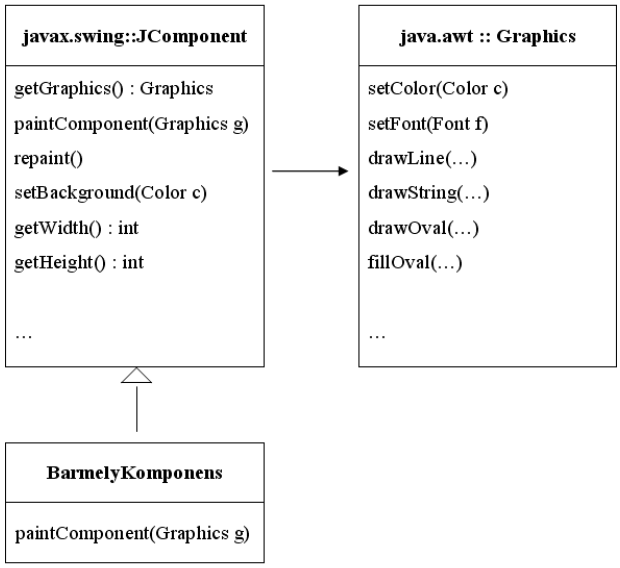
Rajzoláskor a grafikus objektum biztosítja, hogy ne írhassunk a komponensen kívüli területre – azt egyszerűen nem rajzolja ki.

A grafikus obj. továbbadható más objektumnak is, hogy az rajzoljon rá.

A komponens Graphics g objektuma lefedi a komponens teljes felületét. A grafikus objektum bal felső sarkának koordinátái: (0,0). A koordináták pixelben értendők.

Az ablak keretére nem lehet rajzolni. (És a JFrame -re sem!)

RAJZOLÁS SWING FELÜLETEN



NÉHÁNY METÓDUS

Ovális rajzolása

```
drawOval(int x, int y, int width, int height);
```

Kitöltött ovális rajzolása

```
fillOval(int x, int y, int width, int height);
```

Téglalap rajzolása

```
drawRect(int x, int y, int width, int height);
```

Kitöltött téglalap rajzolása

```
fillRect(int x, int y, int width, int height);
```

NÉHÁNY METÓDUS

Szöveg kirajzolása

drawString(String szoveg, int x, int y);

Rajzoló szín beállítása

setColor(Color c);

Pl.: setColor(Color.red);

Pl.: setColor(new Color(int R, int G, int B));

Aktuális rajzoló szín meghatározása

getColor(Color c);

Rajzoló betűtípus beállítása

setFont(Font betutipus);

NÉHÁNY METÓDUS

Kép „rajzolása”

```
drawImage(Image kep, int x, int y, int width, int height,
          ImageObserver mire);
```

Pl.:

```
Image kep; int x = 0, y = 0, szel = 500, mag =500;
kep = new ImageIcon(this.getClass().
          getResource("/kepek/kep.jpg")).getImage();
g.drawImage(kep, x, y, szel, mag, null);
```

STB… → HELP

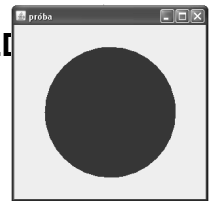
RAJZOT TARTALMAZÓ PROGRAM FELÉPÍTÉSE

1. Saját JFrame

2. Erre: SajatPanel extends JPanel {…}

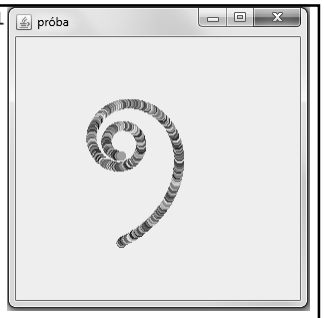
   mert felül kell definiálni a paintComponent(graphics g) metódust

RAJZOLÁS SWING FELÜLETEN – PÉLD

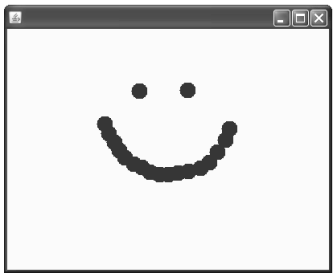



```
import java.awt.Color;
import java.awt.Graphics;

public class RajzPanel extends javax.swing.JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.red);
        int r=100;
        int szelesseg = this.getWidth();
        int magassag = this.getHeight();
        g.fillOval(szelesseg/2-r, magassag/2-r, 2*r, 2*r);
    }
}
```

```
public class RajzPanel extends javax.swing.JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        rajzol(g);
    }

    private void rajzol(Graphics g) {
        int x, y, kpx = 100, kpy = 100, r = 10;
        Color szin;
        double sugar = 100;
        double novekmeny = 0.01;
        double veg = 4 * Math.PI;

        for (double szog = 0; szog < veg; szog += novekmeny) {
            szin = new Color((int) (Math.random() * 256),
                    (int) (Math.random() * 256),
                    (int) (Math.random() * 256));
            g.setColor(szin);
            x = kpx + (int) (sugar / (1 + szog / 2) * Math.sin(szog));
            y = kpy + (int) (sugar / (1 + szog / 2) * Math.cos(szog));
            g.fillOval(x, y, r, r);
        }
    }
```



RAJZOLÁS SWING FELÜLETEN – 3. PÉLDA



Egérkattintásra jelenjen meg a felületen egy piros pötty.

Felépítés:

JFrame → vezérlés

JPanel → esemény + rajzolás

Hogy lehet sok pöttyöt kezelni?

```
public class RajzPanel extends javax.swing.JPanel {

    private List<Potty> pottyok = new ArrayList<>();
```

RAJZOLÁS SWING FELÜLETEN – 3. PÉLDA

```
class Potty {

    private int x, y, r;
    private Color szin = Color.red;

    public Potty(int x, int y, int r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    void rajzol(Graphics g) {
        g.setColor(szin);
        g.fillOval(x-r, y-r, 2*r, 2*r);
    }
```

☹

+ set / get

RAJZOLÁS SWING FELÜLETEN – 3. PÉLDA

A PottyokPanel metódusai:

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    for(Potty p: pottyok){
        p.rajzol(g);
    }
}
```

Fontos:
Ne maradjon le a super hivatkozás!

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    int r = 10;
    pottyok.add(new Potty(evt.getX(),evt.getY(),r));
    this.repaint();
}
```

RAJZOLÁS SWING FELÜLETEN – 4. PÉLDA

Az 1. példát módosítsuk úgy, hogy gombnyomásra véletlenszerűen változzon a körlap színe!

```
public class RajzPanel extends javax.swing.JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        this.rajzol(g);
    }
```

Generálás: Alt+ins. Override Methods, JComponent

RAJZOLÁS SWING FELÜLETEN – 4. PÉLDA

```
private void rajzol(Graphics g) {
    int kozepX = this.getWidth()/2, kozepY = this.getHeight()/2,
        sugar = 100;
    int piros = (int) (Math.random() * 256);
    int zold = (int) (Math.random() * 256);
    int kek = (int) (Math.random() * 256);
    Color szin = new Color(piros, zold, kek);
    g.setColor(szin);
    g.fillOval(kozepX-sugar, kozepY-sugar, 2*sugar, 2*sugar);
}
```

RAJZOLÁS SWING FELÜLETEN – PÉLDA 4.

De ki kényszeríti rajzolásra a panelt?

A rajzGomb másik panelen van.

A frame-n:

```
// Variables declaration - do not modify
private ketPanel.GombPanel gombPanel1;
private ketPanel.RajzPanel rajzPanel1;
// End of variables declaration
public KetPanelesFrame() {
    initComponents();
    gombPanel1.setRajzPanel(rajzPanel1);
}
```

Lehet a konstruktorban az átadás?

RAJZOLÁS SWING FELÜLETEN – PÉLDA 4.

De ki kényszeríti rajzolásra a panelt?

A GombPanelen:

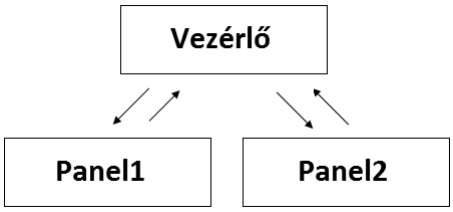
```
private RajzPanel rajzPanel;

public void setRajzPanel(RajzPanel rajzPanel) {
    this.rajzPanel = rajzPanel;
}

private void rajzolGombActionPerformed(java.awt.event.ActionEver
    rajzPanel.repaint();
}
```

RAJZOLÁS SWING FELÜLETEN – PÉLDA 4.

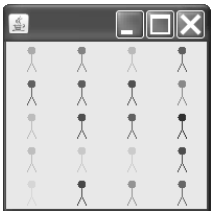
FONTOS MEGJEGYZÉS:

A panelek közötti közvetlen kapcsolat csak ilyen pici feladatok esetén engedhető meg.

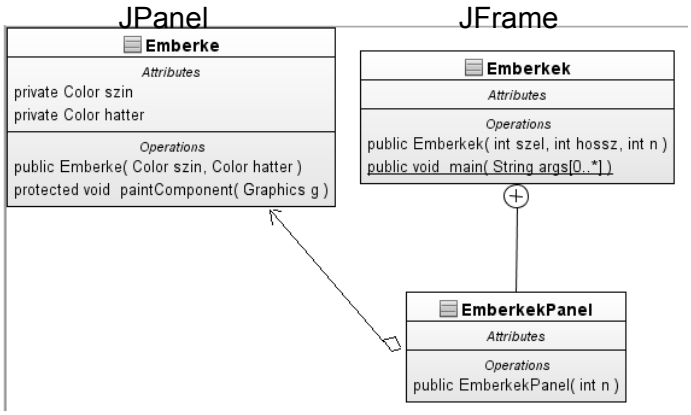
Kicsit is komolyabb feladatok esetén:



RAJZOLÁS SWING FELÜLETEN – 5. PÉLDA



Rajzoljunk „emberkéket” egy swing felületre!



ügyes layout

HF ☺

RAJZOLÁS SWING FELÜLETEN

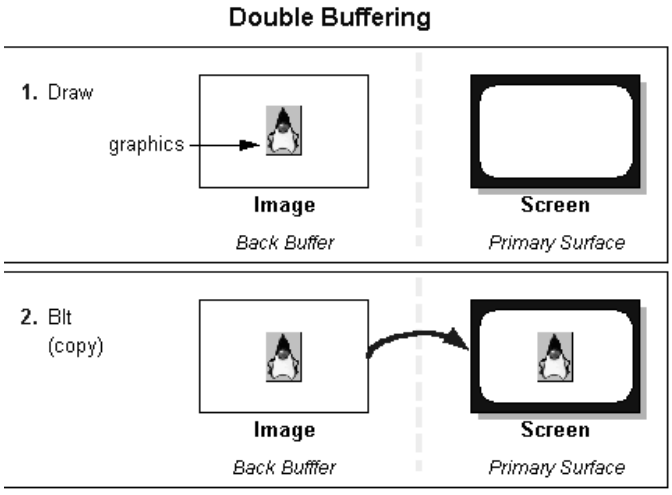
Problémafelvetés – ami Swingben már nem is igazi probléma.

Ha egérmozgatás hatására történik valami (vagy bármilyen más mozgás esetén) előfordulhat, hogy villog a kép.

A probléma lehetséges megoldása a dupla (vagy akár tripla) bufferezés.

RAJZOLÁS SWING FELÜLETEN

Dupla bufferezés:



RAJZOLÁS SWING FELÜLETEN

Vagyis előbb egy „offscrean” képre rajzolunk, majd később ez kerül ki a látható képernyőre.

Általában a villogás csökkentésére szokták használni, vagy olyankor, ha a rajzolás több időt vesz igénybe, mint a monitorfrissítés.

Egy lehetséges megvalósítás (köv. oldal):

(Ez a legegyszerűbb, nem is alkalmazhatjuk minden esetre, a továbbiaknak nézzen utána, ha szüksége van rá.)

A Swing alapértelmezetten használja a dupla bufferezést.

RAJZOLÁS SWING FELÜLETEN

```
    private Image pufferKep;
    private Graphics pufferGrafika;

@Override
protected void paintComponent(Graphics g){
    super.paintComponent(g);
    if(pufferGrafika==null){
        pufferKep=this.createImage(super.getWidth(),super.getHeight());
        pufferGrafika=pufferKep.getGraphics();
        pufferGrafika.clearRect(0, 0, this.getWidth(), this.getHeight());
    }

    g.drawImage(pufferKep, 0, 0, this);
}

  private void formMouseClicked(java.awt.event.MouseEvent evt) {
      pufferGrafika.fillOval(evt.getX()-r/2, evt.getY()-r/2, r, r);
      repaint();
  }
```

RAJZOLÁS SWING FELÜLETEN – 6. PÉLDA

Egérrel kattintva a felületre, a kattintás helyén jelenjen meg egy piros pötty.

Vajon miért nincs kerete?

Mert applet

APPLETEK

Asztali alkalmazások

- Applikációk (eddigi programjaink)

Webes alkalmazások

- Kliens oldalon : **APPLETEK**
- Szerver oldalon : Szervletek

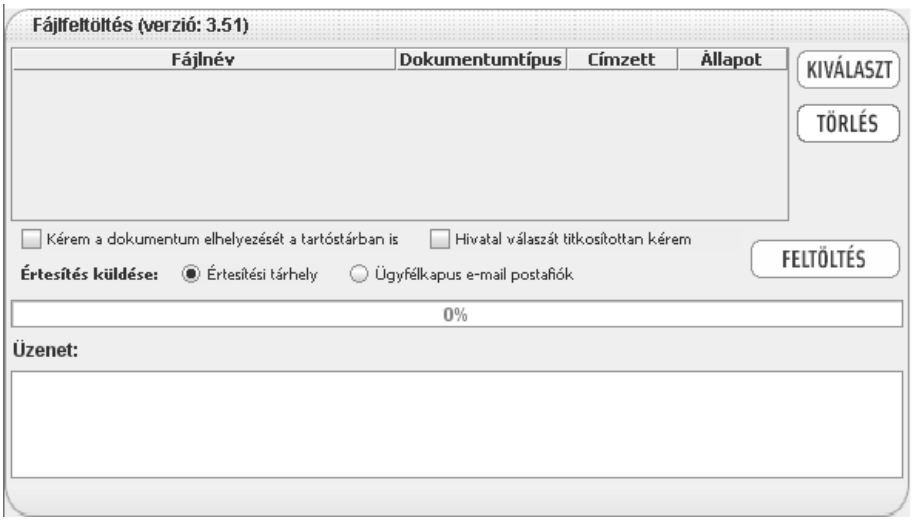
Alkalmazási kör:

- Bonyolultabb programozást igénylő kliens oldali programok.
- Tipikus alkalmazásuk: fájl feltöltés

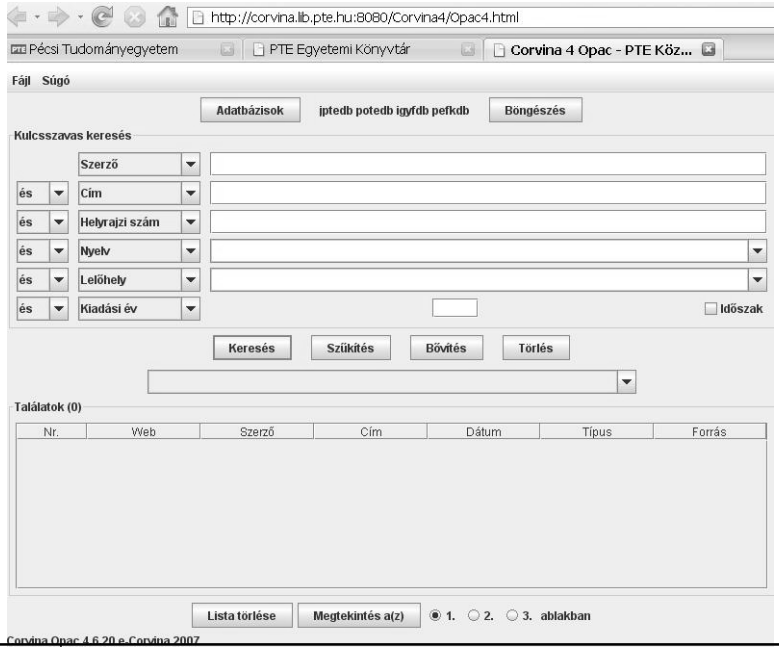
Komolyabb webes alkalmazás: J2EE technológia!

ALKALMAZÁSI PÉLDÁK

Ügyfélkapu:



ALKALMAZÁSI PÉLDÁK



APPLETEK JELLEMZŐI

Java nyelven írt program

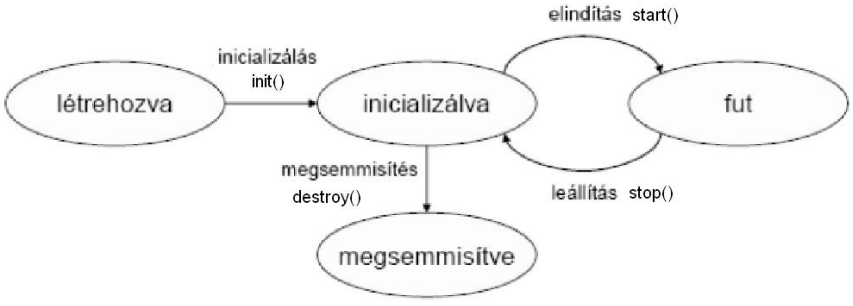
Futtatáshoz böngészőre van szükség (nincs main metódus)

Egy *HTML* oldalba kell beágyazni

Futtatás:

- Böngészőbe épített JVM
- JRE Plugin
- Appletviewer

APPLETEK JELLEMZŐI



Az applet nem ablak!!! Nincs kerete, címe, ikonjai és nem lehet ablakesemény forrása.
Az Applet osztály közvetlen őse a Panel.

APPLETEK JELLEMZŐI

Az appletek a **javax.swing.JApplet** osztályból származnak.

Fontos metódusok:

init() – az applet inicializálásakor hajtódik végre, az applet konstruktorának lefutása után – paraméterek átvétele

start() – az applet elindításakor vagy újraindításakor kerül sorra

stop() – megálláskor fut le

destroy() – az applet megszüntetésekor – erőforrások felszabadítása

paint() – ez felel a rajzolásért

APPLETEK LÉTREHOZÁSA

1. javax.swing csomag importálása

```
import javax.swing.JApplet;
```

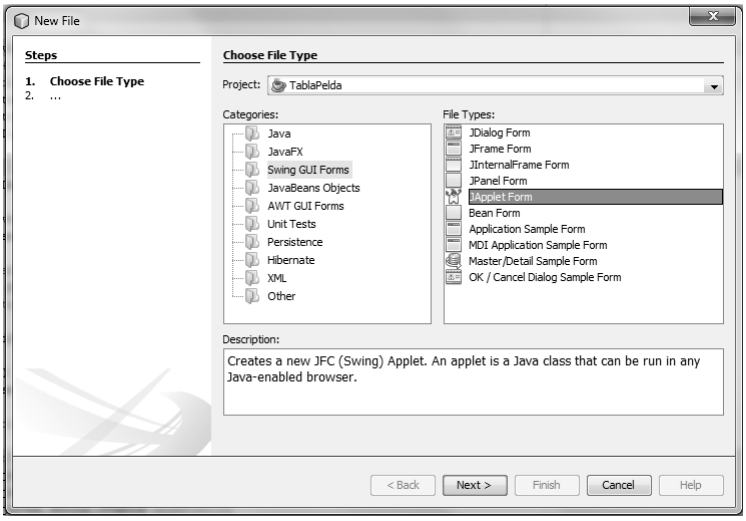
2. saját osztály származtatása az JApplet osztályból

```
public class SajatApplet extends JApplet {
```

3. metódusok megvalósítása

Az 1-2. generálható a NetBeans-ben ☺

APPLETEK LÉTREHOZÁSA



APPLET – PÉLDA

```
public class PottyokJApplet extends javax.swing.JApplet {

    @Override
    public void init() {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)

        /* Create and display the applet */
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {

                public void run() {
                    initComponents();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

APPLET – PÉLDA

```
getContentPane().add(pottyokPanel1, java.awt.BorderLayout.CENTER);
```

A PottyokPanel ugyanaz, mint a korábbi grafikus példában. De bármilyen másik SajatPanel típusú példányt is rárakunk, így egy grafikus alkalmazás egy pillanat alatt átalakítható appletre, csak a panelt nem a frame, hanem az applet felületére kell ráhúzni.

Tehát appletet ugyanúgy hozhatunk létre, mint egy JFrame-t, csak a generálni kívánt típus JApplet Form.

Futtatás: fájlként vagy html-ből

APPLETEK INDÍTÁSA – AZ ELŐZŐ PÉLDÁHOZ GENERÁLT HTML

```
<HTML>
<HEAD>
   <TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>

<H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>

<P>
<APPLET codebase="classes" code="pottyokCsomag/PottyokJApplet.class"
        width=350 height=200></APPLET>
</P>

<HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans IDE</I></FONT>
</BODY>
</HTML>
```

APPLET – MÁSIK PÉLDA – PARAMÉTERÁTADÁS

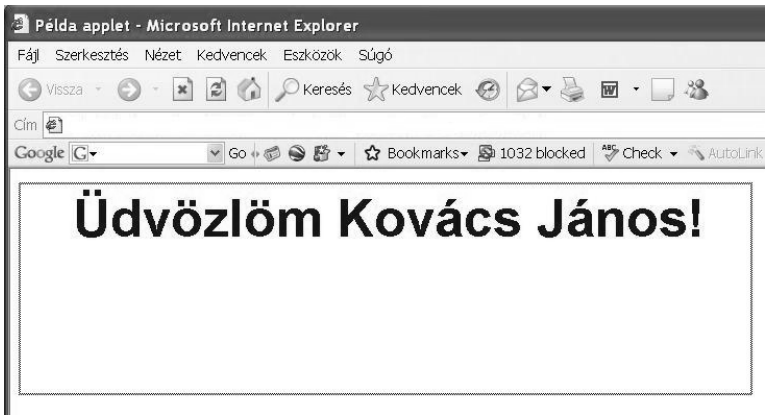
Paraméterátadás html-ből

```
<html>
  <head>
    <title>Példa applet</title>
  </head>
  <body bgcolor = white>

      <applet
        codebase = "file:///d:/"
            code = "Pelda.class"
            width = "700"
            height = "200"
            alt="Hiba van az appletben!"
          >
          <param name = "vezeteknev" value="Kovács"/>
          <param name = "keresztnev" value="János"/>
      </applet>

  </body>
</html>
```

APPLET – MÁSIK PÉLDA – PARAMÉTERÁTADÁS

```
/*
 * Kiírja a html oldalon paraméterként megadott nevet
 **/

import java.awt.*;
import java.applet.*;

public class Pelda extends Applet {

    public void init() {
        setBackground(Color.white);
    }

    public void paint(Graphics g) {

        String vezetek = getParameter("vezeteknev");
        String kereszt = getParameter("keresztnev");

        Font f = new Font("TimesRoman",Font.BOLD,50);

        g.setFont(f);
        g.setColor(Color.blue);
        g.drawString("Üdvözlöm " + vezetek + " " + kereszt + "!", 50,50);
    }
}
```

APPLET – MÁSIK PÉLDA – PARAMÉTERÁTADÁS



APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

Alkalmazás:

- általában korlátlanul használhatja a rendszer erőforrásait
- a helyi rendszerbe telepített Java kódot a környezet megbízhatónak tekinti, futását nem ellenőrzi.
  (Ez persze nem jelenti azt, hogy a program tényleg megbízható, csak azt, hogy a felelősséget a rendszer a program telepítőjére hárítja.)

Applet:

A hálózaton böngészve elég egy óvatlan kattintás az egéren, és a felhasználó számára alig észrevehetően már fut (és gonoszkodik) is egy applet a felhasználó gépén.

VAGY MÉGSEM? $\Rightarrow$ Biztonsági kérdések

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

Hálózatba kapcsolt számítógépeket fenyegető támadások:

Helyi információk feltárása (*disclosure attack*)
A támadó hozzájut a rendszerben tárolt fontos, titkos információkhoz.

Tárolt információk módosítása (*integrity attack*)
A támadó megváltoztathatja a háttértáron, illetve a központi tárban tárolt információkat. Esetleg más, új programokat indíthat el.

Szolgáltatások használhatatlanná tétele (*denial of service attack*)

Felhasználók bosszantása (*annoyance attack*)

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

A támadások elhárításához a Java appletek rendszererőforrások-hoz való hozzáférését kell szigorúan szabályozni, korlátozni.

Védendő erőforrások pl.:
állományrendszer, hálózat, központi tár, be-, kiviteli eszközök, egyéb perifériák, felhasználói környezet, rendszerhívások, rendszerkönyvtárak, stb.

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

Védekezési szintek:

- Nyelvi szint (láthatóság, kivételkezelés, stb.)
- JVM: Az appletek köztes kódra lefordított formáját és nem a forrását töltjük le. ⇒ nem lehetünk biztosak abban, hogy a kódot egy korrekt fordítóprogram állította-e elő, vagy tréfás kedvű számítógépbetyár ⇒ a JVM futtatás előtt ellenőrzi, hogy a kód megfelel-e bizonyos szemantikai tulajdonságoknak.
- Böngészőbe beépített védelem.

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

Megszorítások:

Az appleteknek soha nincs közvetlen hozzáférésük a CPU-hoz és az operációs rendszerhez, ezért a virtuális gép megakadályozhatja fájlok ellopását és vírusok bevitelét.

Az applet nem nyithat meg fájlokat közvetlenül az operációs rendszer rutinjain keresztül, nem hozhat létre hálózati kapcsolatot, s nincs joga más egyéb kockázatos műveletekhez sem. Az applet ezeket a műveleteket csak a Java osztályain keresztül hajtja végre.

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

Megszorítások:

Az appleteknek nincs joguk:

- a felhasználó gépének fájlrendszerét olvasni;
- a felhasználó gépének fájlrendszerét írni;
- a felhasználó gépén lévő fájlokról információt szerezni;
- a felhasználó gépének fájlrendszeréből fájlt törölni;
- néhány kivétellel lekérdezni a rendszer tulajdonságait;
- a kliens valamely hálózati portjára csatlakozni;
- a származási HTTP szervertől különböző gép bármely hálózati portjára csatlakozni;
- könyvtárat vagy DLL-t betölteni;
- más programot vagy szkriptet végrehajtani;
- a virtuális gépet kilépésre kényszeríteni;
- címsor nélküli előugró ablakot nyitni;
- stb.

APPLETEK BIZTONSÁGA (CSAK ÉRINTŐLEGESEN)

De a legbiztonságosabb:

Ha nem muszáj, ne írjunk appletet. ☺