Programozás III

SZÁLAK +

MULTIMÉDIA

PROBLÉMAFELVETÉS

Mit csinál?

```
import java.awt.*;
import java.applet.*;
import java.util.*;

public class PontosIdo extends Applet {

   Calendar naptar = Calendar.getInstance();
   int ora = naptar.get(Calendar.HOUR_OF_DAY);
   int perc = naptar.get(Calendar.MINUTE);
   String akt_ido;
   Font kijelzo;

   public void init() {
       kijelzo = new Font("Times", Font.BOLD,20);
   }

   public void start(){
       int ora, perc, mp;
       while(true){
           naptar = Calendar.getInstance();
           ora = naptar.get(Calendar.HOUR_OF_DAY);
           perc = naptar.get(Calendar.MINUTE);
           mp = naptar.get(Calendar.SECOND);
           akt_ido = String.valueOf(ora)+":"+String.valueOf(perc)
                                              +":"+ String.valueOf(mp);
           System.out.println(akt_ido);
           repaint();
       }
   }

   public void paint(Graphics g) {
       g.setFont(kijelzo);
       g.setColor(Color.blue);
       g.drawString(akt_ido,20,50);
   }
}
```

PROBLÉMAFELVETÉS – RÉSZLET

```
  public void start(){
      int ora, perc, mp;
      while(true){
          naptar = Calendar.getInstance();
          ora = naptar.get(Calendar.HOUR_OF_DAY);
          perc = naptar.get(Calendar.MINUTE);
          mp = naptar.get(Calendar.SECOND);
          akt_ido = String.valueOf(ora)+":"+String.valueOf(perc)
                                              +":"+ String.valueOf(mp);
          System.out.println(akt_ido);
          repaint();
      }
  }

  public void paint(Graphics g) {
      g.setFont(kijelzo);
      g.setColor(Color.blue);
      g.drawString(akt_ido,20,50);
  }
}
```

JAVÍTÁSI ÖTLET

```
public void start(){
    int ora, perc, mp;
    while(true){
        naptar = Calendar.getInstance();
        ora = naptar.get(Calendar.HOUR_OF_DAY);
        perc = naptar.get(Calendar.MINUTE);
        mp = naptar.get(Calendar.SECOND);
        akt_ido = String.valueOf(ora)+":"+String.valueOf(perc)
                                            +":"+ String.valueOf(mp);
        System.out.println(akt_ido);
        for(int i=0; i<1000000; i++){
        };
        repaint();
    }
}

public void paint(Graphics g) {
    g.setFont(kijelzo);
    g.setColor(Color.blue);
    g.drawString(akt_ido,20,50);
}
```

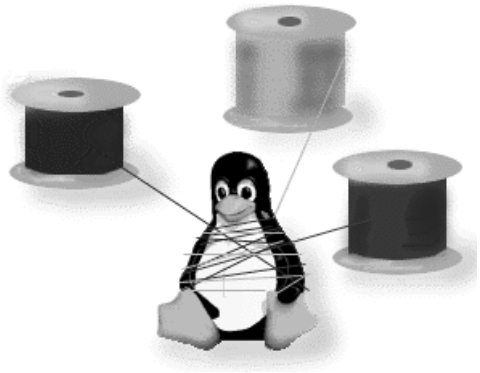
JOBB JAVÍTÁSI ÖTLET

```
public void start(){
    int ora, perc, mp;
    while(true){
        naptar = Calendar.getInstance();
        ora = naptar.get(Calendar.HOUR_OF_DAY);
        perc = naptar.get(Calendar.MINUTE);
        mp = naptar.get(Calendar.SECOND);
        akt_ido = String.valueOf(ora)+":"+String.valueOf(perc)
                                          +":"+ String.valueOf(mp);
        // System.out.println(akt_ido);
        try{
            Thread.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Hiba: " + e.getMessage());
        }
        repaint();
    }
}
```

☹ még mindig nem látszik semmi mert az applet is alszik

MEGOLDÁS

SZÁLKEZELÉS

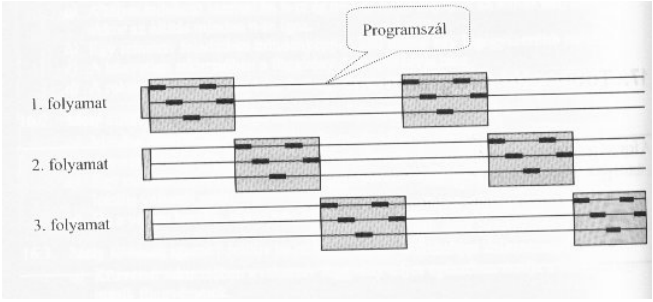


SZÁLKEZELÉS – BEVEZETŐ HASONLAT

Operációs rendszerek:
multiuser-multitasking rendszerek.

Ütemezés:
megadja, hogy a párhuzamosan futó folyamatok közül éppen melyik kapja meg a vezérlést.

Hasonlóan működik egy **többszálú program** is:



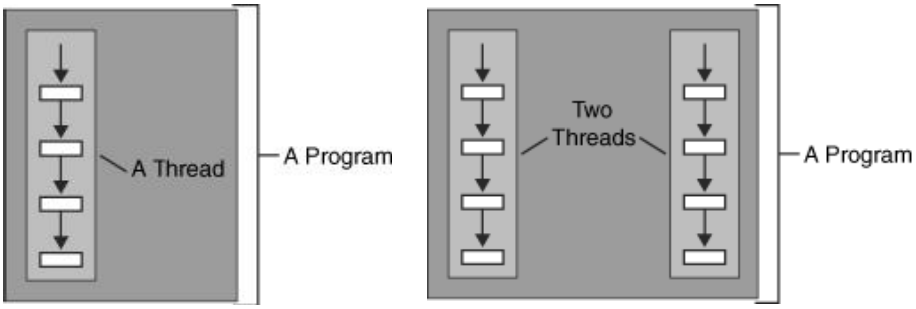
SZÁLKEZELÉS

A szálkezelés alapja a programszál:
A szál (*thread*) egy egyedülálló irányító folyamat a programon belül.

A szál hasonló a soros programhoz:
– van kezdete, végrehajtandó része és befejezése;
– a szál is csak bizonyos időszakaszokban fut.
De a szál maga nem önálló program, nem lehet önállóan futtatni, csak a program részeként fut.

Egy programban több szál futhat „párhuzamosan”.

SZÁLKEZELÉS



A PROGRAMSZÁL ÁLLAPOTAI

A programszál megszületik, működik és meghal. Születésétől haláláig a következő állapotokba juthat:

Új (new): megszületésekor ebbe az állapotba kerül, és ebben van, amíg el nem indítják.

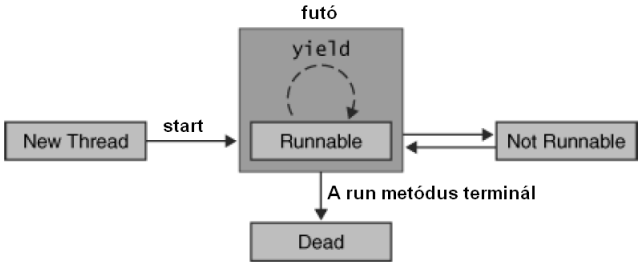
Futtatható (runnable): az ütemező hatáskörébe került:
futó: éppen fut
készenléti: futásra kész, de nem ő van soron.

Blokkolt, felfüggesztett: nem képes futni, pl. valamilyen I/O-ra vagy engedélyezésre vár.

Megszakított: a szál megszakítása blokkolt szálak esetén is azonnal feldolgozódik.

Halott (dead): a szál már nem él, befejezte futását.

A PROGRAMSZÁL ÉLETCIKLUSA



A PROGRAMSZÁL „ÉLŐ" ÁLLAPOTA

1. **futtatható** (runnable) állapot:
   a szál aktív, legfeljebb CPU-ra vár – itt közöljük az elvégzendő feladatot.

2. **blokkolt** (suspended) állapot:
   ilyenkor valami akadályozza a szál futását:

   alvó (sleep) : a szálat el lehet altatni valamennyi időre. Ha ez az időtartam letelik, akkor felébred és fut tovább.

   várakozó (wait): a szál addig vár, amíg fel nem ébresztik. Ha jön az ébresztés (notify), akkor a programszál tovább fut.

ÁLLAPOTÁTMENETEK

új → futtatható (készenléti): *start()* hatására

futó ←→ készenléti: ütemezés hatására, (szálak ütemezése)

futtatható → halott: run() metódus vége

futtatható → megszakított: *interrupt()* hatására

blokkolt → megszakított: *interrupt()* hatására

futtatható (futó) → blokkolt: *wait(), sleep(),* illetve I/O vagy másik szál hatására

blokkolt → futtatható (készenléti): *notify()*
vagy az idő letelte, másik szál befejeződése, I/O vége

SZÁLAK – „ALVÁS-ÉBREDÉS"

Az alatt az idő alatt, amíg a szál alvó állapotban van, a szál nem fut, még akkor sem, ha a processzor erőforrásai elérhetővé válnak.
Miután letelik az alvásra szánt idő, a szál újra futtathatóvá válik: ha a processzor erőforrásai elérhetővé válnak, a szál újra futni kezd.

Minden egyes nem futtatható állapotba kerüléskor egy specifikus és megkülönböztethető *exit* metódus téríti vissza a szálat a futtatható állapotba, De csak megfelelő feltételek teljesülése esetén fog ismét futni.

SZÁLAK – „ALVÁS-ÉBREDÉS”

A nem futtatható állapotból való kilépés feltételei:

• Ha a szál alvó állapotban van, akkor le kell telnie a meghatározott időnek.

• Ha egy szál egy feltételre vár, akkor egy másik objektumnak kell értesítenie a várakozó szálat a feltétel teljesüléséről a *notify* vagy *notifyAll* metódus meghívásával.

• Ha egy szál blokkolva volt egy I/O művelet miatt, az I/O műveletnek be kell fejeződnie.

JAVA PROGRAMSZÁL

A programszál is egy objektum: a Thread osztály leszármazottja.

A Thread osztály a java.lang csomag része, ezért nem kell külön importálni.

Létrehozása:

(1) **Thread** osztály kiterjesztésével vagy

(2) a **Runnable** interfész implementálása + egy Thread példány létrehozása.

Miért van szükség kétfajta megoldásra?

JAVA PROGRAMSZÁL A THREAD KITERJESZTÉSÉVEL

1. A Thread osztály kiterjesztésével létrehozott osztályt **példányosítjuk**.
Ekkor létrejön egy szál, de még nem működik!

2. A futtatáshoz meg kell hívnunk a Thread osztályból örökölt **start()** metódust. Ez inicializálja a szálat, és elindítja annak **run()** metódusát.

3. A **run()** metódus a szál „főprogramja”, itt kap helyet az összes olyan művelet, amelyet a szálnak el kell végeznie.
Szál készítésekor ezt a run() metódust kell átírnunk!!!

JAVA PROGRAMSZÁL A RUNNABLE IMPLEMENTÁLÁSÁVAL

A Runnable interfész egyetlen **run()** metódust tartalmaz, használatára általában akkor van szükség, ha a run metódust tartalmazó osztály nem származhat a Thread osztályból.

Ilyenkor egy, a **Runnable interfész**t megvalósító osztály egy példányát adjuk át a Thread osztály konstruktorának.

Példákon világítjuk meg az eddigieket:

Példa: Hozzunk létre két szálat, és írjuk ki, hogy éppen melyik szál működik.

JAVA PROGRAMSZÁL A THREAD KITERJESZTÉSÉVEL PÉLDA

```
class Szal extends Thread {

    @Override
    public void run() {
        System.out.println("Ez a " + this.getName() + " szál");
    }
}

public class Proba {

    public static void main(String[] args) {
        Szal a, b;
        a = new Szal();
        a.setName("egyik");
        b = new Szal();
        b.setName("másik");
        a.start();
        b.start();
    }
}
```

```
Ez a egyik szál
Ez a másik szál
```

JAVA PROGRAMSZÁL A RUNNABLE IMPLEMENTÁLÁSÁVAL PÉLDA

```
class Szal implements Runnable {

    @Override
    public void run() {
        System.out.println("Ez a " +
                           Thread.currentThread().getName() + " szál");
    }
}

public class Proba {

    public static void main(String[] args) {
        Szal programSzal = new Szal();
        Thread a = new Thread(programSzal);
        a.setName("egyik");
        Thread b = new Thread(programSzal);
        b.setName("másik");
        a.start();
        b.start();
    }
}
```

```
Ez a egyik szál
Ez a másik szál
```

A THREAD OSZTÁLY NÉHÁNY FONTOS METÓDUSA

start()
Elindítja a programszálat. Meghívja a run() metódust.

sleep()
Segítségével egy szálat el lehet altatni a paraméterként megadott ezredmásodpercig. Például:
Thread.sleep(1000) 1 másodpercre elaltatja a szálat.

isAlive()
Visszaadja, hogy a élő-e a szál.

currentThread()
Visszaadja az éppen futó szálobjektumot.

getName()
Visszaadja az éppen futó szálobjektum nevét.

STB… később

SZÁLAK – NÉHÁNY PÉLDA

1. feladat:
Készítsünk konzolos programot:
a) Egy szál segítségével fél másodpercenként (500 ezredmásodperc) írjunk ki egy-egy véletlenül generált mondatot. (Soronként egyet.)
b) Egy másik szál segítségével készítsünk bekezdéseket, vagyis ez a szál véletlen időközönként (1-3 másod-percenként) emeljen sort.

1. FELADAT – MEGOLDÁS

```
public class Szoveg {

    private final static int VARAKOZAS = 500;
    private static final int ALSO = 1000;
    private static final int FELSO = 3000;

    public static void main(String args[]) {
        Szal1 egy = new Szal1(true,VARAKOZAS);
        Szal2 ketto = new Szal2(ALSO, FELSO, true);
        egy.start();
        ketto.start();
    }
}
```

Létrehozzuk és elindítjuk a szálakat

```
class Szal1 extends Thread {
    private boolean fut;
    private int varakozas;

    public Szal1(boolean fut, int varakozas) {...4 lines }

    @Override
    public void run() {
        String alany[] = {"kutya", "macska", "tanár", "diak"};
        String allitmany[] = {"ugat", "nyávog", "tanít", "tanul","alszik", "bulizik"};
        String jelzo[] = {"", "okos", "ügyes", "álmos", "buta", "bátor"};

        String nevelo;
        int i1, i2, i3;

        while (fut) {
            i1 = (int) (Math.random() * jelzo.length);
            i2 = (int) (Math.random() * alany.length);
            i3 = (int) (Math.random() * allitmany.length);

            nevelo = "A(z)";
            System.out.println(nevelo + " " + jelzo[i1] + " "
                    + alany[i2] + " " + allitmany[i3] + ".");
            try {
                sleep(varakozas);
            } catch (InterruptedException ex) {
                Logger.getLogger(Szal1.class.getName()).log(Level.SEVERE, null, ex);
            }
        }   }
```

1. FELADAT – MEGOLDÁS (FOLYT.)

```
class Szal2 extends Thread {

    int ido;
    int also , felso;
    boolean fut;

    public Szal2(int also, int felso, boolean fut) {
        this.also = also;
        this.felso = felso;
        this.fut = fut;
    }

    @Override
    public void run() {
        while (fut) {
            ido = (int) (Math.random() * (felso-also + 1) + also);
            System.out.println();
            try {
                sleep(ido);
            } catch (InterruptedException ex) {
                Logger.getLogger(Szal2.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

1. FELADAT – EREDMÉNY

```
--------------------Configura
A(z) bátor diak ugat.

A(z) álmos diak alszik.
A(z) ügyes kutya bulizik.
A(z) okos tanár alszik.

A(z) buta kutya ugat.
A(z) okos macska ugat.
A(z) bátor tanár alszik.
A(z) ügyes macska ugat.

A(z)  diak bulizik.
A(z) buta tanár nyávog.
A(z) buta diak tanít.
A(z) ügyes diak ugat.
A(z) álmos tanár nyávog.

A(z) bátor macska tanít.
A(z) ügyes tanár nyávog.
A(z)  kutya ugat.
A(z) bátor tanár ugat.
A(z) buta diak nyávog.

Process interrupted by user.
```

HF:

1. „Rendes” névelők megoldása.
2. Külső fájlból beolvasott szókincs alkalmazása.
3. „Mintha” most gépelnénk – azaz betűnként írja ki.
4. Stb.

SZÁLAK – NÉHÁNY PÉLDA

2. feladat:
Javítsuk ki a bevezető példa applet-jét.
(Vagyis írjunk egy olyan appletet, amely másodpercenként kiírja az aktuális időt.)

2. FELADAT – MEGOLDÁS

Ez egy grafikus alkalmazás, ezért a lépések:

1. JApplet alapú vezérlő osztály
2. JPanel ráhelyezése, DE…

a panelnek most sok mindent kell tudnia.

JPanel alapú osztályt hozunk létre

A panelen valami mozog  szálkezelés kell

2. FELADAT – MEGOLDÁS (1. RÉSZ)

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

    @Override
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {

                public void run() {
                    initComponents();
                    idoPanel1.szalinditas();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @SuppressWarnings("unchecked")
    Generated Code
    // Variables declaration - do not modify
    private idocsomag.IdoPanel idoPanel1;
    // End of variables declaration
}
```

2. FELADAT – MEGOLDÁS (2. RÉSZ)

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.text.DateFormat;
import java.util.Date;


public class IdoPanel extends javax.swing.JPanel
                      implements Runnable {

    private DateFormat datumForma;
    private String aktIdo="";
    private Font kijelzo;
    private Thread szal;
    private boolean online = true;

    public IdoPanel() {
        initComponents();
        kijelzo = new Font("Times", Font.BOLD, 20);
        System.out.println(aktIdo);
    }
```

2. FELADAT – MEGOLDÁS (3. RÉSZ)

```
public void szalinditas() {

    if (szal == null) {
        szal = new Thread(this);
        szal.start();
    }
}

public void run() {
    int ido = 1000;
    while (online) {
        datumForma = DateFormat.getTimeInstance();
        aktIdo = datumForma.format(new Date());
        try {
            Thread.sleep(ido);
        } catch (InterruptedException e) {
            System.out.println("Hiba: " + e.getMessage());
        }
        repaint();
    }
}
```

Kétszer nem lehet elindítani ugyanazt a szálat!!!

2. FELADAT – MEGOLDÁS (4. RÉSZ)

```
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setFont(kijelzo);
    g.setColor(Color.blue);
    g.drawString("Idő: " + akt_ido, 20, 50);

}

@SuppressWarnings("unchecked")
Generated Code


// Variables declaration - do not modify
// End of variables declaration

}
```

2. FELADAT – MEGOLDÁS ALKALMAZÁSKÉNT

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

    public IdoFrame() {
        initComponents();
        this.setLocationRelativeTo(null);
    }
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            IdoFrame frame = new IdoFrame();
            frame.setVisible(true);
            frame.indit();
        }
    });
}
            private void indit() {
                 idoPanel1.szalinditas();
            }
        }
```

2. FELADAT – MEGOLDÁS (DÁTUM MÁSKÉPP)

A dátum szebb kiíratáshoz szükséges részletek:

```
import java.util.Date;
import java.text.DateFormat;

    DateFormat datumforma;
    String akt_ido ="";

datumforma = DateFormat.getTimeInstance();
akt_ido=datumforma.format(new Date());
```

MEGJEGYZÉSEK:
1. A DateFormat osztály kiterjesztéseként definiált SimpleDateFormat jóval több formázási lehetőséget enged.

2. Vigyázat! A Date többi konstruktora és sok metódusa elavult!! (deprecated)

SZÁLAK – ANIMÁCIÓ

Szálkezeléssel jól megvalósítható a késleltetés:

Thread.sleep()

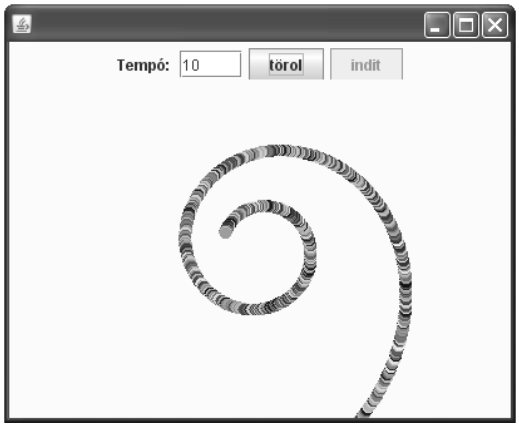
a paraméterében megadott ezredmásorpercnyi időre elaltatja a szálat -> késleltetés.

A metódus megköveteli, hogy kivételkezelést alkalmazzunk, tehát try{}catch(){} blokkba kell tenni.

Úgy ébred fel, hogy interruptedExceptiont dob.

SZÁLAK – NÉHÁNY PÉLDA

3. feladat:
Készítsünk grafikus alkalmazást, amely szál segítségével lassan (vagyis szabad szemmel is követhetően) elkezd rajzolni egy csigavonalat.



3. FELADAT – MEGOLDÁS

Ez is grafikus alkalmazás, ezért a lépések:

1. JFrame alapú vezérlő osztály
2. JPanel ráhelyezése, DE…

a panelnek most sok mindent kell tudnia.

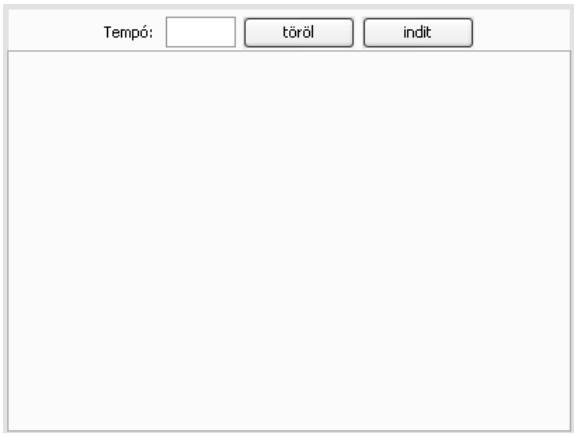
JPanel alapú osztályt hozunk létre

A panelen valami mozog szálkezelés kell

3. FELADAT – MEGOLDÁS

CsigaFrame:

Egy lehetséges megoldás: a frame-n két panel van:



Lehet „gyári” panel, de kicsit is érdekesebb feladok esetén jobb a saját.

(Flow Layout)

saját panel

3. FELADAT – MEGOLDÁS

CsigaFrame (most a vezérlő panel „gyári”):

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

    private boolean rajzol = false;

    public CsigaFrame() {...}

    @SuppressWarnings("unchecked")
    Generated Code

    private void rajzolGombActionPerformed(java.awt.event.ActionEver

    private void inditGombActionPerformed(java.awt.event.ActionEvent

    public static void main(String args[]) {...}
```

```
public class CsigaPanel extends javax.swing.JPanel
                        implements Runnable {

    private List<Potty> pottyok = new ArrayList<>();
    private boolean rajzol = false;
    // csak akkor kell, ha rajzolni is és törölni is akarunk
    private int ido, kpx, kpy;
    private Thread szal;

    public CsigaPanel() {...}

    public void indit(int ido) {...}

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

    @Override
    public void run() {...}

    void ujPotty(int sugar, double szog) {...}

    void pottyTorles() {...}

    public boolean isRajzol() {...}

    public void setRajzol(boolean rajzol) {...}
```

3. FELADAT – MEGOLDÁS

CsigaFrame: indit

```
private void inditGombActionPerformed(java.awt.event.ActionEv
    int ido;
    try {
        ido = Integer.parseInt(tempoBe.getText());
        if (ido > 0) {
            csigaPanel1.indit(ido);
            inditGomb.setEnabled(false);
        }else{
            throw new Exception();
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,
                                "írjon be egy pozitív számot");
        tempoBe.grabFocus();
    }
}
```

3. FELADAT – MEGOLDÁS

CsigaFrame: töröl vagy rajzol

```
private void rajzolGombActionPerformed(java.awt.event.Acti

    csigaPanel1.setRajzol(rajzol);
    if (rajzol) {
        rajzolGomb.setText("torol");
    } else {
        rajzolGomb.setText("rajzol");
    }
    rajzol = !rajzol;
}
```

3. FELADAT – MEGOLDÁS

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

A Potty osztály rajzol(Graphics g) metódusa kirajzolja az adott középpontú, sugarú és színű körlapot.

```
public class CsigaPanel extends javax.swing.JPanel
                        implements Runnable {

    private List<Potty> pottyok = new ArrayList<>();
    private boolean rajzol = false;
    // csak akkor kell, ha rajzolni is és törölni is akarunk
    private int ido, kpx, kpy;
    private Thread szal;

    public CsigaPanel() {
        initComponents();
    }

    public void indit(int ido) {
        this.ido = ido;
        kpx = this.getWidth() / 2;
        kpy = this.getHeight() / 2;
        szal = null;
        szal = new Thread(this);
        szal.start();
        this.rajzol = true;
    }
```

3. FELADAT – MEGOLDÁS

```
@Override
public void run() {
    double szog = 0, inc = 0.01;
    int sugar = 200;
    // a csigavonal sugara
    while (!szal.isInterrupted()) {
        if (rajzol) {
            ujPotty(sugar, szog);
            szog += inc;
        } else {
            pottyTorles();
            szog -= inc;
        }
        try {
            Thread.sleep(ido);
        } catch (InterruptedException ex) {
            Logger.getLogger(CsigaPanel.class.getName()).log(Level.SE
        }
        repaint();
    }
}
```

3. FELADAT – MEGOLDÁS

```
void ujPotty(int sugar, double szog) {
    int x, y, r = 5;
    Color szin;

    szin = new Color((int) (Math.random() * 256),
            (int) (Math.random() * 256),
            (int) (Math.random() * 256));

    x = kpx + (int) (sugar / (1 + szog / 2) * Math.sin(szog));
    y = kpy + (int) (sugar / (1 + szog / 2) * Math.cos(szog));
    pottyok.add(new Potty(x, y, r, szin));
}

void pottyTorles() {
    if (!pottyok.isEmpty()) {
        pottyok.remove(pottyok.size() - 1);
    }
}
```

HF.: Kipróbálni más-más függvényekkel.

3. FELADAT – PROBLÉMA

Hibásan működik. ☹

```
run:
Exception in thread "AWT-EventQueue-0" java.util.ConcurrentModificationException
        at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:819)
        at java.util.ArrayList$Itr.next(ArrayList.java:791)
        at csiga.CsigaPanel.paintComponent(CsigaPanel.java:39)
        at javax.swing.JComponent.paint(JComponent.java:1054)
        at javax.swing.JComponent.paintToOffscreen(JComponent.java:5221)
```

Hibaforrás: a paintComponent() for ciklusa.

Ok: „szálösszeakadás” – konkurens módosítási probléma.

Megoldás: ArrayList helyett

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

KITÉRŐ: MEDDIG MŰKÖDIK?

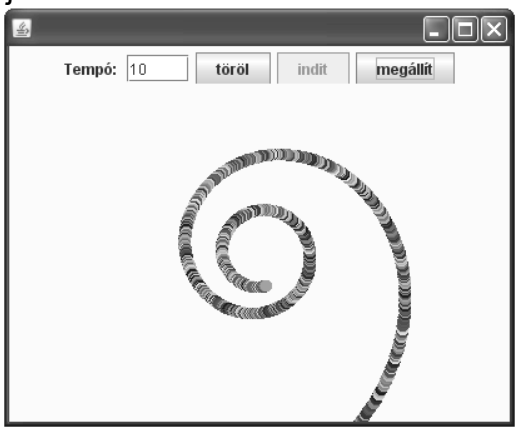
```
public void run() {
    double szog = 0, inc = 0.01;
    int sugar = 200;

//      while (!szal.isInterrupted()) {
    while(szog <= 2*Math.PI){
        if (rajzol) {
            ujPotty(sugar, szog);
            szog += inc;
            try {
                Thread.sleep(ido);
```

2π -ig rajzol, utána befejeződik a run(), vagyis szal.isAlive() = false.

Vagyis nem mindig „végtelen” ciklus a szál run() metódusa!!

3. FELADAT – BŐVÍTÉS

Bővítsük az előző feladatot! Egy gombnyomás hatására álljon meg a mozgás, majd újra megnyomva a gombot, induljon újra.



3. FELADAT – MEGOLDÁS

```
private void megallitGombActionPerformed(java.awt.event.Acti
    megy = !megy;
    csigaPanel1.valtas(megy);
    if(megy) megallitGomb.setText("megállít");
    else megallitGomb.setText("folytat");
}
```

A gombnyomás esemény hatására a szálnak várakoznia kell, majd újabb gombnyomás hatására újraindulnia.

A gombnyomást össze kell hangolni a szállal.

szinkronizálni kell

SZÁLAK VÁRAKOZTATÁSA

wait():
Az a szál, amelyik meghívja a wait metódust, blokkolódik, és addig vár, amíg egy bizonyos feltétel nem teljesül. Amikor jelzést kap, hogy újra futhat, megpróbálja folytatni a futást.

notify():
Egy várakozó szálat továbbenged. Amikor a várakozó szál megkapja az üzenetet, megpróbál továbbhaladni.

Néha szükség lehet arra, hogy az összes várakozó szál feléledjen. Erre szolgál a notifyAll() metódus.

SZÁLAK SZINKRONIZÁCIÓJA

Egy program (applet) több szálat is létrehozhat.

– ezek lehetnek egymástól függetlenek

– de előfordulhat, hogy több szál ugyanazokkal az adatokkal, objektumokkal dolgozik. Ilyenkor szükség lehet arra, hogy az egyik programszál bevárja a másikat. Vagyis szükség lehet a szálak **szinkronizációjára**.

Ha egynél több szál fér hozzá egy adott változóhoz, és az egyikük felül is írhatja, akkor az összes szál hozzáférését össze kell hangolnunk (szinkronizálnunk).

SZÁLAK SZINKRONIZÁCIÓJA

A Java szálmodelljében a szinkronizáció az ún. **monitor**ok segítségével valósul meg.

– Minden objektumhoz (példányhoz) tartozik egy monitor.

– A monitort **egyszerre egy szál birtokolhatja**.

– Ha egyszerre több szál is igényt tart a monitorra, a többi szál várakozni kényszerül, futása felfüggesztődik.

– Ha a monitort birtokló szál elengedi a monitort, a monitort igénylő többi szál verseng a monitorért. A szálak közül az ütemező választja ki azt, amelyik a monitort megkapja.

3. FELADAT – MEGOLDÁS (FOLYT.)

A gombnyomásnak és a mozgásnak szinkronban kell lennie: a gombnyomástól függően a szálnak vagy várakoznia kell (wait()), vagy futhat (notify()).

Ezért szinkronizálni kell a megfelelő metódusokat:

```
synchronized void valtas(boolean megy) {
    this.megy = megy;
    if(megy) this.notify();
}
```

ahol

```
private boolean megy = true;
```

de persze, később változhat az értéke.

3. FELADAT – MEGOLDÁS (FOLYT.)

```
public void run() {
    double szog = 0, inc = 0.01;
    int sugar = 200;

      while (!szal.isInterrupted()) {
        if(!megy){
            synchronized(this){
                try {
                    this.wait();
                } catch (InterruptedExceptior
                    Logger.getLogger(CsigaPar
                }
            }
        }
        if (rajzol) {
            ujPotty(sugar, szog);
            szog += inc;
         ...
```

Szinkronizált blokk.

A blokkba való belépéskor a blokkot végrehajtó szál megszerzi a megadott objektum monitorát, a blokkból való kilépéskor elengedi azt.

3. FELADAT – MEGOLDÁS (FOLYT.)

Természetesen így is lehet:

```
                while (!szal.isInterrupted()) {
                  varakozas();
                  if (rajzol) {
                      ujPotty(sugar, szog);
                            ...

synchronized void varakozas() {
    if (!megy) {
        try {
            this.wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(CsigaPanel.class.getName()).log(Level.SEVERE,
        }
    }
}
```

3. FELADAT – MEGOLDÁS (FOLYT.)

Megjegyzés:

A konkurencia probléma is megoldható szinkronizálással. (Az ujPotty(), pottyTorles() metódusokat kell szinkronizálni.)

De jobb (és egyszerűbb) a

CopyOnWriteArrayList

használata.

SZÁLAK – MÓDOSÍTÓK

A volatile módosító használata:

Az olyan változókat kell ellátni ezzel a módosítóval, amelyeket egy másik párhuzamosan futó process vagy szál is használ. A módosítónak az lesz a hatása, hogy a fordító minden hivatkozásnál újra beolvassa a memóriából a változót (még akkor is, ha egy korábbi hivatkozás eredményeként már benne van egy regiszterben). Ezzel lehet biztosítani, hogy ha közben egy process megváltoztatta a változót, akkor ezt a változtatást az aktuális program szál azonnal figyelembe vegye.

SZÁLAK – NÉHÁNY PÉLDA

4. feladat:

Kitérő: kis multimédia

EGY CSÖPP MULTIMÉDIA – HANG

```
package hangcsomag;

import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;

public class HangFrame extends javax.swing.JFrame {

     private AudioClip klipp;
     private boolean megy = false;
     public HangFrame() {
        initComponents();
        try {
            URL cim = this.getClass().getResource("/zene/poirot_zene.wav");
            klipp = Applet.newAudioClip(cim);
        } catch (Exception ex) {
            System.out.println("Hiba: " + ex.getMessage());
        }
     }
```

EGY CSÖPP MULTIMÉDIA – HANG

```
private AudioClip klipp;
private boolean megy = false;
public HangFrame() {
    initComponents();
    try {
        klipp = Applet.newAudioClip(new URL("file:///C:/temp/poirot_zene.wav"));
    } catch (Exception ex) {
        System.out.println("Hiba: " + ex.getMessage());
    }
}
    private void zeneGombActionPerformed(java.awt.event.ActionEvent evt) {
        if(!megy){
            klipp.play();
            zeneGomb.setText("leállít");
            megy = true;
        }
        else{
            klipp.stop();
            zeneGomb.setText("elindít");
            megy = false;
        }
    }
```

EGY CSÖPP MULTIMÉDIA – HANG

```
try {
    klipp = Applet.newAudioClip(new URL("file:///C:/temp/poirot_zene.wav"));
} catch (Exception ex) {
    System.out.println("Hiba: " + ex.getMessage());
}
```

A megoldás hibája?

```
try {
//     klipp = Applet.newAudioClip(new URL("file:///C:/temp/poirot_zene.wav"));
    URL cim = this.getClass().getResource("/zene/poirot_zene.wav");
    klipp = Applet.newAudioClip(cim);
} catch (Exception ex) {
    System.out.println("Hiba: " + ex.getMessage());
}
```

A zene mappa helye: src könyvtár, hangcsomag mellett.

EGY CSÖPP MULTIMÉDIA – KÉP

```
import java.awt.*;
import java.applet.*;
import java.net.URL;

public class KepProba extends Applet {

 Image kep;

 public void paint(Graphics g) {

   try{
     // kep = getImage(new URL("http://www.sg.hu/kep/2008_11/wallerloco.jpg"));
        kep = getImage(getCodeBase(),"kep.jpg");
        g.drawImage(kep,0,0,this);
   }
   catch(Exception e){
       System.out.println("Hiba: " + e.getMessage());
       g.drawString("Nincs ilyen kép", 100,100);
   }
  }
}
```

jpg, gif, png

NetBeans-ben működik, parancsmódban nem. ☹

???

EGY CSÖPP MULTIMÉDIA – KÉP

Magyarázat:

– NetBeans-ben működik, létrehoz egy applet.policy fájlt, melynek tartalma:
grant {
permission java.security.AllPermission;
};

További magyarázat:

http://www-personal.umich.edu/~lsiden/tutorials/signed-applet/signed-applet.html

EGY CSÖPP MULTIMÉDIA – KÉP

```
private Image kep;
public KepPanel() {
    initComponents();
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    kep = new ImageIcon(this.getClass().getResource("/kepek/kep.jpg")).
                                                                getImage();
    int kezdox = 0, kezdoy = 0,
        szelesseg = this.getWidth(),
        magassag = this.getHeight();
    g.drawImage(kep, kezdox, kezdoy, szelesseg, magassag, this);
}
```

Image kep = new ImageIcon(cim).getImage()

EGY CSÖPP MULTIMÉDIA – KÉP ÉS HANG

```
public class HangKepPanel extends javax.swing.JPanel
                          implements Runnable{

    private AudioClip klipp;
    private Thread szal;
    private Image kep[], aktualisKep;
    private MediaTracker mtrack;
    private int kepszam = 7;
    private long ido = 4000;
    private boolean online = true;

    public HangKepPanel() {
        initComponents();
        feltolt();
        lejatszas();
    }
```

EGY CSÖPP MULTIMÉDIA – KÉP ÉS HANG (2)

```
public void szalInditas(){
    if(szal == null){
        szal = new Thread(this);
        szal.start();
    }
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawImage(aktualisKep, 0, 0, this.getWidth(),
                                   this.getHeight(), this);
}

private void lejatszas() {
    URL cim = this.getClass().getResource("/zene/poirot_zene.wav");
    klipp = Applet.newAudioClip(cim);
    if(klipp!=null) klipp.play();
}
```

EGY CSÖPP MULTIMÉDIA – KÉP ÉS HANG (3)

```
private void feltolt(){
    mtrack = new MediaTracker(this);
    kep = new Image[kepszam];
    URL cim;
    for(int i=0; i<kep.length; i++){
        cim = this.getClass().
                getResource("/kepek/"+"kep" + (i+1) + ".jpg");
        kep[i] = new ImageIcon(cim).getImage();
        mtrack.addImage(kep[i],i);
    }
    try{
        mtrack.waitForAll();
    } catch(InterruptedException e){
        System.out.println("Hiba: " + e.getMessage());
    }
}
```

java.awt.MediaTracker – segítségével egyszerre több kép betöltését tudjuk nyomonkövetni.

Ez most megvárja, amíg az összes betöltődik.

EGY CSÖPP MULTIMÉDIA – KÉP ÉS HANG (4)

```
public void run(){
    int db=0;
    while(online){
        aktualisKep = kep[db];
        try{
            Thread.sleep(ido);
            this.repaint();
        }
        catch(Exception e){
            System.out.println("Hiba: " + e.getMessage());
        }
        db++;
        if(db>=kep.length) db=0;
    }
}
```

SOK MULTIMÉDIA

javax.swing…

javax.sound… – önálló feldolgozás

Pl.: bemutatandó vizsgafeladatok

Most vissza a szálakhoz, de előtte egy kis kitérő ínyenceknek.

ÜGYESEBB ZENEHASZNÁLAT

Tavalyi vizsgafeladatból: .ogg kiterjeszésű zenéket is jól lehet használni (jóval kisebb méret).

Ingyenes online konverter:
http://audio.online-convert.com/convert-to-ogg

A megoldáshoz külső library-k kellenek, ezért célszerű maven projektként kezelni a feladatot, és a pom.xml fájlban leírni a szükséges függőségeket.

Még egy segítség:
https://stackoverflow.com/questions/9752972/how-to-add-an-extra-source-directory-for-maven-to-compile-and-include-in-the-bui

```
import helper.Global;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.newdawn.slick.Music;
import org.newdawn.slick.SlickException;
import org.newdawn.slick.Sound;

/**
 * Audio HashMap-eket tartalmazó osztály.
 * @author Balázs
 */
public class AudioPlayer {

    public static Map<String, Sound> soundMap = new HashMap<>();
    public static Map<String, Music> musicMap = new HashMap<>();

    /**
     * HashMap-et használva az audiohoz kellő értékpárokat elhelyezem.
     */
    public static void load() {

        //Csak az english path megfelelo neki nem szereti a hosszú magánhangzókat a library..
        try {
            soundMap.put(Global.MENU_SOUND_STRING, new Sound(Global.MENU_SOUND_STRING_PATH));
            musicMap.put(Global.MENU_MUSIC_STRING, new Music(Global.MENU_MUSIC_STRING_PATH));
        } catch (SlickException ex) {
            Logger.getLogger(AudioPlayer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static Music getMusic(String key) {
        return musicMap.get(key);
    }

    public static Sound getSound(String key) {
        return soundMap.get(key);
    }
}
```

TAKARÉKOSKODÁS

A gépen egyidejűleg futó szálak osztoznak egymás között a processzoron ⇒ fontos, hogy az egyes szálak ne pazarolják értelmetlenül a processzor-időt.

Különösen az appletnél fontos, hogy a szálak szabályosan befejeződjenek.

Ha a böngésző másik oldalra vált, akkor az applet pihenésbe kezd, ha visszatérünk, akkor ismét feléled. DE

Ez csak az appletre vonatkozik, a szálakra NEM. Erről a programozónak kell gondoskodnia.

TAKARÉKOSKODÁS

Leggyakoribb megoldás:

Ez nem a Thread stop() metódusa! Az elavult!

```
    public void stop(){
        if(szal != null) szal = null;
    }
}   // applet vége
```

Ekkor visszatérve az appletbe, a start ismét új szálat indít.

Megoldható a „jegelés” is, de elég bonyolultan – a régi, egyszerű megoldásokat a JDK 1.2 óta nem tekintik biztonságosnak – ezek deprecated metódusok.

SZÁLBIZTOSSÁG

Egy osztály akkor szálbiztos, ha több szálból hozzáférve is helyesen viselkedik, függetlenül az ütemezéstől vagy attól, hogy a futásidejű környezet hogyan fűzi keresztbe az említett szálak végrehajtását, és nincs szükség további összehangolásra vagy más egyeztetésre a hívó kód részéről.

Brian Goetz: Párhuzamos Java-programozás a gyakorlatban

SZÁLBIZTOSSÁG ÉS A SWING

Szinte minden grafikus eszközkészletet, így a Swinget is egyszálas alrendszerként valósítottak meg, vagyis minden grafikus tevékenység egyetlen szálra van felfűzve. Ez az úgynevezett esemény szál.
Feladata: a komponensek és események kezelése.

A swing komponenseinek legtöbb metódusa nem szálbiztos, azaz nincs felkészítve a konkurens hozzáférés lehetőségére. Ha nem az eseményszálból hívjuk meg ezeket, akkor hiba történhet!

SZÁLBIZTOSSÁG ÉS A SWING

A GUI-val kapcsolatos funkciókat a java.awt.EventQueue vagy a javax.swing.SwingUtilities invokeLater() vagy invokeAndWait() metódusával az esemény szálban hajtjuk végre. Pl.:

```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new OraFrame().setVisible(true);
        }
    });
}
```

SZÁLBIZTOSSÁG ÉS A SWING

Ha egy hosszadalmas feladat fut az eseményszálon, akkor a felhasználó még a „mégsem” gombra sem kattinthat, hiszen a rendszer ennek eseménykezelőjét is csak a hosszú feladat befejezését követően hívja meg.

Fontos, hogy az eseményszálon elindított feladatok a lehető leghamarabb visszaadják a vezérlést a szálnak.

Ha tehát egy hosszan futó feladatot szeretnénk kezdeményezni, akkor azt egy másik szálban kell megtennünk.

SZINKRONIZÁCIÓ + EGYEBEK

Néha a szinkronizáció sem elég.

A swing nem szálbiztos ⇒ csak az AWT Event szál tudja rendesen kezelni a swing objektumokat.
Ha ebbe „bele akarunk nyúlni”, azaz az esemény-figyelőn kívül akarunk módosítani valamit, akkor célszerű a SwingUtilities.invokeLater() metódust használni. Ez „besorolja” a mi hivatkozásunkat az eseménysorba, és azonnal sorraveszi.

SWINGUTILITIES.INVOKELATER – PÉLDA

Például a saját szálban SwingUtilities.invokeLater-ben hívjuk meg a revalidate() vagy repaint() metódust:

```
public void run() {

        ...

    SwingUtilities.invokeLater(new Runnable(){
        public void run() {
            repaint();
        }
    });
```

Ügyesebb megoldás: szálvezérlő osztály használata – majd később.

SZÁLAK, IDŐZÍTÉS

Egyszerűbb esetekben nincs szükség saját szálak kezelésére.

Ha pl. a programban egy feladatot többször kell elvégezni, akkor érdemes a *java.util.Timer* osztályt alkalmazni.
A *Timer* osztály időzítéses feladatoknál is hasznos lehet.

Időzítés: Timer, TimerTask

HatarIdo.class

Tennivalo.class

```
import java.util.Timer;
import java.util.TimerTask;

public class HatarIdo {
   Timer idozito;

   public HatarIdo(int seconds) {
       idozito = new Timer();
       idozito.schedule(new Tennivalo(idozito), seconds*1000);
   }

   public static void main(String args[]) {
       new HatarIdo(5);
       System.out.println("Elkezdődött a munka.");
   }
}

 class Tennivalo extends TimerTask {
   Timer idozito;
   public Tennivalo(Timer idozito){
       this.idozito = idozito;
   }
   public void run() {
       System.out.println("Eltelt az idő!");
       idozito.cancel(); //A timer szál megszüntetése
   }
}
```

Időzítés: Timer, TimerTask

```
import java.util.Timer;
import java.util.TimerTask;

// Megoldás belső osztállyal

public class HatarIdo2 {
   Timer idozito;

   public HatarIdo2(int seconds) {
       idozito = new Timer();
       idozito.schedule(new Tennivalo(), seconds*1000);
   }

   class Tennivalo extends TimerTask {
       public void run() {
           System.out.println("Eltelt az idő!");
           idozito.cancel(); //Az idozito szál megszüntetése
       }
   }

   public static void main(String args[]) {
       new HatarIdo(5);
       System.out.println("Elkezdődött a munka.");
   }
}
```

HatarIdo2.class

HatarIdo2$Tennivalo.class

Időzítés: Timer, TimerTask

```
import java.util.Timer;
import java.util.TimerTask;

// Megoldás beágyazott osztállyal

public class HatarIdo3 {
   Timer idozito;

   public HatarIdo3(int seconds) {
       idozito = new Timer();
       idozito.schedule(new TimerTask(){
           public void run() {
               System.out.println("Eltelt az idő!");
               idozito.cancel(); //Az időzítő szál megszüntetése
           }}, seconds*1000);
   }

   public static void main(String args[]) {
       new HatarIdo(5);
       System.out.println("Elkezdődött a munka.");
   }
}
```

HatarIdo3.class

HatarIdo3$1.class

Időzítés: Timer, TimerTask

Egy másik lehetőség az ütemezésre, hogy az indítási időpontot adjuk meg. Pl. a következő kód 23:01-re ütemezi a végrehajtást:

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.HOUR_OF_DAY, 23);
calendar.set(Calendar.MINUTE, 1);
calendar.set(Calendar.SECOND, 0);
Date ido = calendar.getTime();
idozito = new Timer();
idozito.schedule(new Tennivalo(), ido);
```

Gondolja végig, hogy az előző változatok közül melyik alakítható át így. HF.: Hogyan alakítható át a többi?

Időzítés: Timer, TimerTask – ismételt futtatás

```
import java.util.Timer;
import java.util.TimerTask;

public class Proba {

   Timer idozito;

   public Proba(int db) {
       idozito = new Timer();
       idozito.schedule(new Feladat(),
                        0, //kezdeti késleltetés
                        db*1000); //ismétlési ráta
   }

   class Feladat extends TimerTask {
       int szamlalo = 10;
       public void run() {
           if (szamlalo > 0) {
           System.out.println("Itt vagyok!");
           szamlalo--;
           } else {
           System.out.println("Lejárt az idő!");
           //timer.cancel(); // Nem szükséges
                             // mert van System.exit is.
           System.exit(0); // Leállítja a szálat
                           // (és minden mást)
           }
       }
   }
   public static void main(String args[]) {
       new Proba(1);
   System.out.println("Munka ütemezve.");
   }
}
```

Időzítés: Timer, TimerTask – ismételt futtatás – részlet

```
Timer idozito;

public Proba(int db) {
    idozito = new Timer();
    idozito.schedule(new Feladat(),
                     0, //kezdeti késleltetés
                     db*1000); //ismétlési ráta
}

class Feladat extends TimerTask {
    int szamlalo = 10;
    public void run() {
        if (szamlalo > 0) {
        System.out.println("Itt vagyok!");
        szamlalo--;
        } else {
        System.out.println("Lejárt az idő!");
        //timer.cancel(); // Nem szükséges
                          // mert van System.exit is.
        System.exit(0); // Leállítja a szálat
                        // (és minden mást)
        }
    }
}
```

Egyéb lehetőségek: HF