Programozás III

SZÁLAK +

MULTIMÉDIA 2.

SZÁLAK (ismétlés)

Programszál:
A Thread osztály egy példánya.

Létrehozása:

– A Thread osztály leszármaztatása

– A Runnable interface implementálása + az implementált példány segítségével létrehozott Thread példány (kompozíció)

Szál indítása: start()

futása: run()

SZÁLAK (ismétlés)

Szál indítása: start()

futása: run()

alvása: sleep(idő)

várakozása: wait()

felébresztése: notify()

Több szál együttes felébresztése: notifyAll()

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.

PÉLDA

Gombnyomás hatására a szálnak vagy várakoznia kell (wait()), vagy futhat (notify()). A kettőnek azonban szinkronban kell lennie.

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.

PÉLDA (FOLYT.)

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

      while (!szal.isInterrupted()) {
        if(!megy){
            synchronized(this){
                try {
                    wait();
                } catch (InterruptedException
                    Logger.getLogger(CsigaPan
                }
            }
        ...
       }
```

Szinkronizált blokk.

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

PÉLDA (FOLYT.)

Természetesen így is lehet:

```
while (!szal.isInterrupted()) {
  varakozas();
  ...
}
```

```
synchronized void varakozas() {
    if (!megy) {
        try {
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(CsigaPanel.class.getName()).lo
        }
    }
}
```

PÉLDA (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.

MÉG EGY PÉLDA

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

```
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 szál-biztos, 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