Programozás III

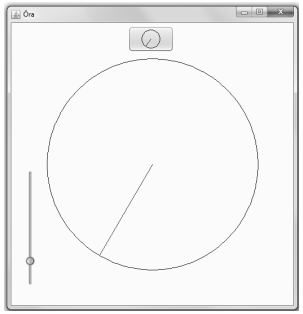
SZÁL ÉS EGYÉB PÉLDÁK

KORÁBBI PÉLDÁKKAL KAPCSOLATOS ÉSZREVÉTELEK

```
public class Gomb extends JButton{

    private Vezerlo vezerlo;

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        if(vezerlo != null) {
            vezerlo.rajzolas(g, this.getWidth(), this.getHeight());
        }
    }

    public void setVezerlo(Vezerlo vezerlo) {
        this.vezerlo = vezerlo;
    }
}
```

KORÁBBI PÉLDÁKKAL KAPCSOLATOS ÉSZREVÉTELEK

„Órás” példa



Hogyan kerülhet a gombra is egy óra?

Saját gomb osztály

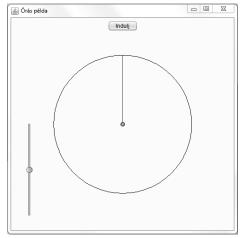
A gombnyomás hatására elindul/megáll (a gombon lévő óra is ☺)

KORÁBBI PÉLDÁKKAL KAPCSOLATOS ÉSZREVÉTELEK

„Órás” példa – másik megoldási lehetőség

Az Ora maga is lehet komponens:

(Ilyenkor akár saját eseményt is rendelhetünk hozzá.)



```
public class Ora extends JComponent implements Runnable{

    private int kx, ky;
        ...

    public Ora() {
    }

    public Ora(Color keretSzin, Color mutatoSzin, double szog,
               boolean fut, long ido) {

        this.keretSzin = keretSzin;
        this.mutatoSzin = mutatoSzin;
        this.szog = szog;

        this.fut = fut;
        this.ido = ido;
    }

    public void beallit(int balx, int baly, int szelesseg, int magassag,
                        double arany) {
        this.setSize(szelesseg, magassag);
        this.setLocation(balx, baly);
        kx = szelesseg/2;
        ky = magassag/2;
        this.r = (int) (arany*((kx < ky) ? kx : ky));
    }
```

Ráhúzható-e a panelre?

Igen, mert bean, azaz van üres konstruktora.

KORÁBBI PÉLDÁKKAL KAPCSOLATOS ÉSZREVÉTELEK

OraPanel osztályban:

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

    private Ora ora;
    private int ido;
    private Thread szal;

    public OraPanel() {
        initComponents();
        // Design módban is beállítható.
        this.setLayout(new BorderLayout());
    }
```

Ebben az osztályban nem kell paintComponent

FONTOS: Csak akkor látszódik, ha a panel BorderLayout!

```
private void formAncestorAdded(javax.swing.event.AncestorEvent evt) {
    this.oraInditas();
}
```

KORÁBBI PÉLDÁKKAL KAPCSOLATOS ÉSZREVÉTELEK

Ora osztályban:

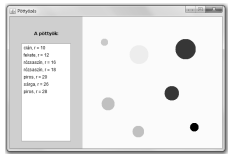
```
@Override
protected void paintComponent(Graphics grphcs) {
    super.paintComponent(grphcs);
    rajzol(grphcs);
}

public void rajzol(Graphics g){
    g.setColor(keretSzin);
    g.drawOval(kx-r, ky-r, 2*r, 2*r);
    g.setColor(mutatoSzin);
    g.drawLine(kx, ky, (int) (kx + r*Math.cos(szog)),
                       (int) (ky - r*Math.sin(szog)));
}
```

```
@Override
public void run() {
    while (fut) {
        ...
        this.repaint();
        ...
    }
}
```

OraPanel osztályban:

```
private void oraInditas() {
    double szog = Math.PI / 2;
    Color keretSzin = Color.green;
    Color mutatoSzin = Color.red;
    ido = 1000;
    // A komponens méretéhez képest milyen arányú legyen az óra átmérője.
    double arany = 0.9;

    ora = new Ora(keretSzin, mutatoSzin, szog, true, ido);
    //   this.repaint();

    ora.setNovekmeny(-Math.PI / 6);

    // Rátesszük a panelre az órát.
    this.add(ora);

    // Megadjuk a komponens méreteit és a rajzolási arányt.
    ora.beallit(0, 0, this.getWidth(), this.getHeight(), arany);

    // A szálindítás az Ora osztályban is megírható, akkor itt csak hívni kell.
    if (szal == null) {
        szal = new Thread(ora);
        szal.start();
    }
}
```

MÁSIK GRAFIKUS FELADATTAL KAPCSOLATOS KITÉRŐ



Hogyan lehet névvel együtt kezelni a színeket?

(Más feladatban is érdekes lehet, pl. comboBox feltöltése.)

1. Két tömb (színek, elnevezések) – így ne! nagyon fapados ☹
2. Szin osztály + lista
3. Hashmap
4. Enum

MÁSIK GRAFIKUS FELADATTAL KAPCSOLATOS KITÉRŐ

Hashmap

```
private void szinekComboItemStateChanged(java.awt.event.ItemEvent evt) {
    if(szinek.containsKey(szinekCombo.getSelectedItem())){
        szin = szinek.get(szinekCombo.getSelectedItem());
    }
}
```

A konkrét feladat (szerintem) nem igényli a hashmap alkalmazását, de nagyon jó példa arra, hogy általában hogyan lehet használni. Ez a szerkezet ugyanis széles körben alkalmazható, **fontos szerkezet**.

MÁSIK GRAFIKUS FELADATTAL KAPCSOLATOS KITÉRŐ

Hashmap

```
private Map<String, Color> szinek = new HashMap<>();

private void szinekFeltolt(){
    szinek.put("fekete", Color.BLACK);
    szinek.put("kék",Color.BLUE);
          …

   for (Map.Entry<String, Color> entry : szinek.entrySet()) {
       szinekCombo.addItem(entry.getKey());
   }
}
```

public static interface **Entry<K,V>**
A map entry (key-value pair). The Map.entrySet method returns a collection-view of the map, whose elements are of this class. …

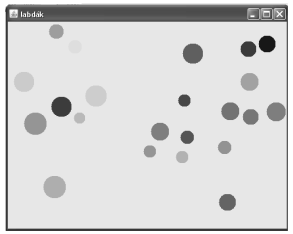
MÁSIK GRAFIKUS FELADATTAL KAPCSOLATOS KITÉRŐ

Enum:

```
public enum Szin {

    fekete(Color.BLACK),
    kék(Color.BLUE),
    cián(Color.CYAN),
    szürke(Color.GRAY),
    zöld(Color.GREEN),
    lila(Color.MAGENTA),
    narancs(Color.ORANGE),
    rózsaszín(Color.PINK),
    piros(Color.RED),
    sárga(Color.YELLOW);

    private Color color;

    private Szin(Color color) {
        this.color = color;
    }

    public Color getColor() {
        return color;
    }
}
```

PÉLDA

Hozzunk létre egy grafikus alkalmazást, amelynek felületén labdák pattognak!



Megoldás:

Labda osztály: extends Thread

+ List<Labda> labdak; – hol deklaráljuk ezt a listát?

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

Megoldás: Szinkronizáció

Csakhogy ez sajnos néha kevés ☹

Meg kell várni, amíg teljesen befejezi a rajzolást, és csak utána szabad törölni.

⬇

Célszerű létrehozni egy szálvezérlő osztályt.

+

ArrayList<> helyett CopyOnWriteArrayList<>

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

A példa folytatása: a labdák automatikusan keletkezzenek, majd próbáljuk „levadászni” őket.

Probléma: Egy idő után összeakadnak a szálak (ConcurrentModificationException)

```
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 javax.swing.JComponent.paint(JComponent.java:1054)
        at javax.swing.JComponent.paintToOffscreen(JComponent.java:5221)
```

Oka: a paintComponent() metódusban ki akarjuk rajzolni a lista összes elemét, de közben már a törlés metódus esetleg kitörölte a kirajzolandó elemet.

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

```
public class SzalVezerlo {

    private List<Labda> labdak = new CopyOnWriteArrayList<>();
```

Hogyan érhető el, hogy automatikusan keletkezzenek?

Pl. a panel is szálként viselkedik ⇒ implements Runnable()

A labdák beszúrásának hívása a panel run() metódusában.

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

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

         private Thread szal;

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

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

„Levadászás”:

Labda-ban:

```
boolean elkaptak(int x, int y) {
    if(Math.sqrt((kx-x)*(kx-x)+(ky-y)*(ky-y))<=r) return true;
    else return false;
}
```

Panel-ben:

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    if(szalVezerlo.torol(evt.getX(), evt.getY())) {
        this.repaint();
    }
}
```

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

Kitérő: Hol indítsuk el a panel-szálat, azaz hol hívjuk meg a Panel szalInditas() metódusát?

Ha a Panel konstruktorában hívnánk, akkor már a tervezési fázisban is fogná a gépet.

A Frame indit() metódusában vagy a Panel AncestorAdded eseményének hatására.

```
labdaPanel1.szalInditas();
```

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

SzalVezerlo-ben:

```
public synchronized void torol(Labda labda) {
    if (labda != null) {
        labda.setFut(false);
    }
    labdak.remove(labda);
}

public synchronized boolean torol(int x, int y) {
    for (Labda labda : labdak) {
        if (labda.elkaptak(x, y)) {
            torol(labda);
            return true;
        }
    }
    return false;
}
```

A törlés nagyon kényes művelet, ha nem lépünk ki azonnal, elszállhat.

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

Kitérő: lehet így is, de jobb az előző

```
synchronized void torol(int x, int y) {
    int torlendo = -1;
    for (int i = 0; i < labdak.size(); i++) {
        if (labdak.get(i).elkaptak(x, y)) {
            torlendo = i;
            break;
        }
    }
    if (torlendo >= 0) {
        labdak.remove(torlendo);
    }
}
```

A kitérő oka: a törlés mindig nagyon kényes művelet, oda kell rá figyelni.

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

Általában: amikor több szál osztozik megváltoztatható adatokon, akkor minden olyan szálat, amely írja vagy olvassa az adatokat, szinkronizálni kell.

Ha a szálakat CopyOnWriteList gyűjteményben kezeljük, akkor elmaradhat a listában lévő szálak közötti szinkronizáció, de ez a listafajta elég költséges, így meggondolandó a használata.

EGY GYAKORI PROBLÉMA ÉS MEGOLDÁSA

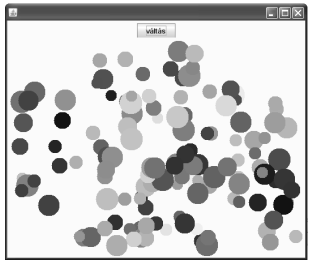
A SzalVezerlo esetleges további metódusai:

```
        public synchronized void osszesTorol(){
            for(Labda labda : labdak){
                labda.setFut(false);
            }
            labdak.clear();
        }
    public synchronized void osszSebessegValtas(int ido) {
        for (Labda labda : labdak) {
            labda.setIdo(ido);
        //    sebessegValtas(labda,ido);
        }
    }
public synchronized void sebessegValtas(Labda labda, int ido) {
    labda.setIdo(ido);
}
```

stb…

ÁLLATORVOSI LÓ

Csak hogy minél több dologról szó eshessen, bővítsük a feladatot:



Rakjunk fel egy gombot, amelynek hatására egyszerre megállítható vagy újraindítható az összes labda.

Hogyan marad átméretezéskor is középen a gomb?

ÁLLATORVOSI LÓ

VezerloPanel-ben:

```
private void valtGombActionPerformed(java.awt.event.Acti
    szalVezerlo.mukodesValtas();
}
```

SzalVezerlo-ben:

```
public synchronized void mukodesValtas() {
    for (Labda labda : labdak) {
        labda.mukodesValtas();
    }
}
```

Labda-ban: ld. a diasorozat elején tárgyalt várakoztatást.

ÁLLATORVOSI LÓ

Másik kitérő: A panelek felrakása.

Mi történik, ha ez a módosítás sorrendje?

1. Leszedjük a frame-ről az eredeti LabdaPanel-t.
2. Kivesszük a LabdaPanel konstruktorából a szalVezerlo példányosítását.
3. Felrakjuk a frame-re a VezerloPanel-t is és a LabdaPanel-t is.

Módosításkor célszerű kikommentezni a korábbi „kényes” hivatkozásokat, majd a panelek ismételt elhelyezése után újra aktívvá tenni ott, ahol kell.

ÁLLATORVOSI LÓ

Hogyan oldható meg, hogy mindkét panel ugyanazt a szalVezerlo példányt használja?

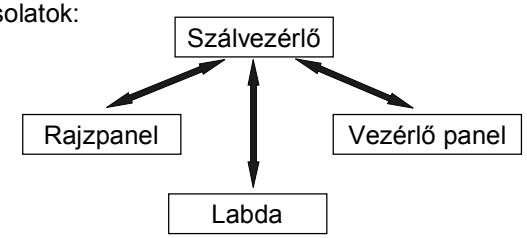
```
private SzalVezerlo szalVezerlo;
public FoFrame() {
    initComponents();

    szalVezerlo = new SzalVezerlo();
    labdaPanel1.setSzalVezerlo(szalVezerlo);
    vezerloPanel1.setSzalVezerlo(szalVezerlo);
}
```

(Természetesen ekkor egyik panelben sem szabad példányosítani a SzalVezerlo-t.)

Kitérő: miért nem a panelek konstruktorában adjuk át a szálat?

AZ ELŐZŐEK SZEMLÉLTETÉSE

Kapcsolatok:



Elvileg a labdának szüksége lenne a rajzpanelre, de nem muszáj közvetlenül elérnie, rábízhatja a frissítést a szál-vezérlőre. Jó, ha minél függetlenebbek az osztályok.

Nem biztos, hogy mindig minden kapcsolat kölcsönös!!

ÁLLATORVOSI LÓ

További módosítás: Fárasztó kattintgatni, most a törlést is a program végezze automatikusan.

Vagyis két szál kell:

1. beszúr egy újabb labdát;
2. töröl egyet.

Ez az úgynevezett Termel – Fogyaszt modell

ÁLLATORVOSI LÓ

A Termel feladata:

Véletlen időközönként létrehoz egy-egy labdát, amelyet a szalVezerlo beszur() metódusa berak a kirajzolandó labdák közé, és el is indítja a mozgását.

A labdák csak a panel felületére rajzolhatók, ezért bár véletlen helyen jönnek létre, de a panel határain belül. Ehhez viszont ismerni kell a panel méreteit, és nyilván a panel repaint() metódusát tudjuk meghívni.

```
public Termel(LabdaPanel labdaPanel, SzalVezerlo szalVezerlo) {
    this.labdaPanel = labdaPanel;
    this.szalVezerlo = szalVezerlo;
}
```

ÁLLATORVOSI LÓ

Panel szerepe:

A LabdaPanel szerepe csak a kirajzolás, de mivel továbbra is a SzalVezerlo osztály rajzol() metódusát használja, ezért szükség van a setSzalVezerlo() metódusra is.

A labdák létrehozásáért a Termel osztály a felelős,

a törlésükért a Fogyaszt osztály.

ÁLLATORVOSI LÓ

Fogyaszt feladata:

Véletlenszerűen kiválasztani egy törlendő labdát, és kitöröltetni a szalVezerlo-vel.

```
while (fut) {
    ido = (int) (Math.random() * (felsoIdo-alsoIdo+1) + alsoIdo);
    index = (int) (Math.random()*szalVezerlo.meret());
    szalVezerlo.torol(index);
    ...
```

ÁLLATORVOSI LÓ

SzalVezerlo:

```
public synchronized void torol(int index){
    Labda labda = labdak.elementAt(index);
    torol(labda);
}
```

Mi a baj? Esetleg akkor is törölni akar, ha nincs egy labda sem.

ÁLLATORVOSI LÓ

A Frame:

```
private SzalVezerlo szalVezerlo;
private Termel termel;
private Fogyaszt fogyaszt;
public FoFrame() {
    initComponents();
    szalVezerlo = new SzalVezerlo();
    labdaPanel1.setSzalVezerlo(szalVezerlo);
}

private void termelesInditas(){
    termel = new Termel(labdaPanel1, szalVezerlo);
    termel.start();
    fogyaszt = new Fogyaszt(labdaPanel1, szalVezerlo);
    fogyaszt.start();
}
```

termelesInditas() hívása: a frame indit() metódusában.

ÁLLATORVOSI LÓ

Javítás:

```
public int meret() {
    return labdak.size();
}
```

```
meret = szalVezerlo.meret();
if (meret > 0) {
    index = (int) (Math.random() * meret);
    szalVezerlo.torol(index);
}
```

Ügyesebb megoldás lenne várakoztatni a szálat, amíg nincs törlendő.

Ezt a változatot most csak konzolos alkalmazásként beszéljük meg, de előbb befejezzük ezt a feladatot.

TERMEL – FOGYASZT MODELL

Egyszerű konzolos alkalmazásként beszéljük meg.

Feladat:
Van egy raktár, amelybe egy termelő időnként beszállít, ha van még hely a raktárban, egy fogyasztó pedig kivesz árut, ha van kivehető áru.

Elvileg megoldhatjuk úgy, mint az előbb, vagyis a méretek figyelésével, de ez esetben fölöslegesen fut a szál akkor is, ha nem tud berakni vagy kivenni a raktárból.

Ilyenkor inkább várakoztatni kell a megfelelő szálat.

```
public class Termelo extends Thread {

    private Raktar raktar;
    private boolean fut = true;

    public Termelo(Raktar raktar) {
        this.raktar = raktar;
    }

    @Override
    public void run() {
        int szorzo = 100;
        while (fut) {
            try {
                Thread.sleep((int) ((Math.random() * szorzo)));
                raktar.betesz();
            } catch (InterruptedException ex) {
                Logger.getLogger(Termelo.class.getName()).log(Level.
            }
        }
    }
}
```

TERMEL – FOGYASZT MODELL

Raktar

```
public class Raktar {

    private int maxKapacitas;
    private int aktMennyiseg;

    public Raktar(int maxKapacitas) {
        this.maxKapacitas = maxKapacitas;
    }

private void kiir() {
    System.out.println("Aktualis mennyiség " + aktMennyiseg);
}
```

```
public class Fogyaszto extends Thread {

    private Raktar raktar;
    private boolean fut = true;

    public Fogyaszto(Raktar raktar) {
        this.raktar = raktar;
    }

    @Override
    public void run() {
        int szorzo = 100;
        while (fut) {
            try {
                Thread.sleep((int) ((Math.random() * szorzo)));
                raktar.kivesz();
            } catch (InterruptedException ex) {
                Logger.getLogger(Fogyaszto.class.getName()).log(Level.
            }

        }
    }
}
```

TERMEL – FOGYASZT MODELL

Raktar

```
public synchronized void betesz() {

    if (maxKapacitas <= aktMennyiseg) {
        try {
            System.out.println("termelo var");
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Raktar.class.getName()).log(Level
        }
    }
    aktMennyiseg++;
    notify();
    kiir();
}
```

TERMEL – FOGYASZT MODELL

Raktar

```
public synchronized void kivesz() {

    if (aktMennyiseg <= 0) {
        try {
            System.out.println("fogyasztó var");
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Raktar.class.getName()).log(Level.
        }
    }
    aktMennyiseg--;
    notify();
    kiir();
}
```

TERMEL – FOGYASZT MODELL

Működés:
Minden objektumhoz tartozik egy ún. *wait-várakozási sor.*
A wait() hatására a szál bekerül ebbe.
A notify() hatására az egyik várakozó kikerül belőle.

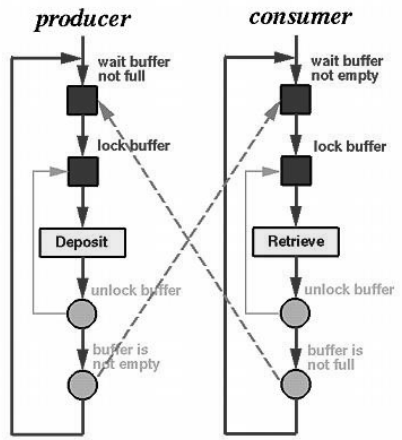
A wait() es notify() hívások csak olyan kódrészben szerepelhetnek, amelyek ugyanazon az objektumon szinkronizáltak.

TERMEL – FOGYASZT MODELL

Main:

```
public class Main {

  private static Termelo termelo;
  private static Raktar raktar;
  private static Fogyaszto fogyaszto;

    public static void main(String[] args) {
        raktar = new Raktar(10);
        fogyaszto = new Fogyaszto (raktar);
        termelo = new Termelo(raktar);
        termelo.start();
        fogyaszto.start();
    }
}
```

TERMEL – FOGYASZT MODELL



https://javarevisited.blogspot.hu/2013/12/inter-thread-communication-in-java-wait-notify-example.html?m=1

SZÁLAKKAL KAPCSOLATOS NÉHÁNY TOVÁBBI FOGALOM

Join:
public final void **join**() throws InterruptedException

Ez a metódus mindaddig vár, amíg az a szál, amelyikre meghívtuk, be nem fejeződik. Neve onnan származik, hogy a hívó szál addig várakozik, amíg a specifikált szál nem *csatlakozik (join)* hozzá.

http://java-latte.blogspot.hu/2014/10/java-thread-join-example-with-explanation.html

SZÁLAKKAL KAPCSOLATOS NÉHÁNY TOVÁBBI FOGALOM

Deadlock:
többszálú rendszerekben könnyen előfordulhat, hogy keresztben egymásra várakozó szálak miatt a program végrehajtása végtelen időre elakad, azaz ún.deadlock lép fel. A deadlock egy bonyolult hiba, amit nehéz megkeresni, mivel indeterminisztikusan lép fel, s több mint két szálat is érinthet.

http://tutorials.jenkov.com/java-concurrency/deadlock.html

http://tutorials.jenkov.com/java-concurrency/deadlock-prevention.html

SZÁLAKKAL KAPCSOLATOS NÉHÁNY TOVÁBBI FOGALOM

Lock:
A szinkronizálás indirekt módon enged/nem enged működni egy szálat, a lock/unlock direkt mechanizmus.

http://winterbe.com/posts/2015/04/30/java8-concurrency-tutorial-synchronized-locks-examples/

http://tutorials.jenkov.com/java-concurrency/locks.html

TERMEL – FOGYASZT MODELL

HF: Az előző labdás példa ilyen átalakítása.

Jónak tűnő anyag:

http://www.theorphys.elte.hu/fizinf/HaloAdat/tananyag/java/A_SZALAK/index.html

Néhány egyéb:

http://java-latte.blogspot.hu/2013/08/thread-communication-with-wait-notify.html

Néhány mintapélda:

előadás-mintapéldák: szalak.zip (Forrás: Zidarics Zoltán)
futtatás: run file (mert minden osztályban van main())