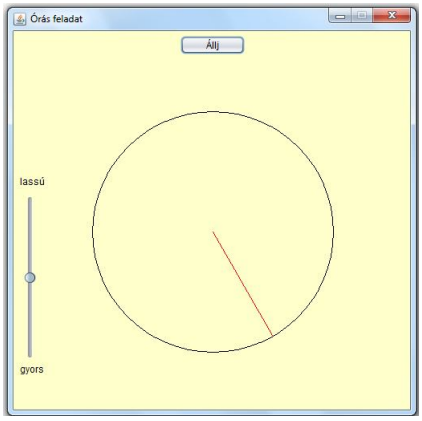
GRAFIKA, SZÁLKEZELÉS

Feladat:
Készítsen egy 500x500-as grafikus alkalmazást egy analóg óra szimulálására!



a) Először készítse el a másodpercmutatót. – ezt most egy vonal helyettesítse, amely 60 helyett 12 lépésben járja be a kört. (Másodpercenként 5 másodpercnyit ugrik.) A mutató a program indulásakor automatikusan elindul, vagyis még nem a gombnyomás hatására.

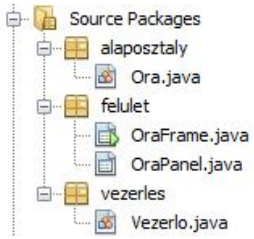
b) Az előző feladatot alakítsa át úgy, hogy egérkattintás hatására változzon meg az órajárás iránya.

d) Most úgy alakítsa át, hogy a mutatót gombnyomás hatására lehessen elindítani (vagyis amíg nem nyomtuk meg a gombot, addig áll az óra). Újabb gombnyomás hatására álljon meg, majd újabb hatására induljon el, stb. Figyeljen a gomb feliratára!

f) Oldja meg, hogy a képen látható módon, egy csúszka segítségével lehessen változtatni a mutató sebességét.

Egy lehetséges megoldás:

A program alapja egy `JFrame` (`OraFrame`), ezen lesz egy saját `JPanel` (`OraPanel` osztály), és erre kerül az `Ora` osztály egy példánya.



Mivel az MVC mintát (vagy legalábbis ahhoz nagyon hasonló elvet) szeretnénk követni, ezért az osztályokat igyekszünk függetleníteni egymástól, és úgy megírni a projektet, hogy azok csak egy vezérlőn keresztül tudjanak kommunikálni egymással. Ezért szükségünk lesz még egy `Vezerlo` osztályra is.

Kezdjük az `Ora` osztállyal! Mivel a mutató mozog, ezért ezt szálként kell definiálnunk, a `rajzolas()` metódusa pedig megadja, hogy hogyan kell kirajzolni az órát, esetünkben egy körvonalat és az óramutatót.

A kezdőértékeket vagy a konstruktoron vagy settereken keresztül tudjuk beállítani, vagy persze az is lehet, hogy egy részét a konstruktorban, egy részét pedig setterek segítségével. Most mindent a konstruktorban adok meg, de ebben az esetben is meg kell írnunk (nyilván átgondoltan) a `set()`és `get()` metódusokat. Erre egy esetleges későbbi módosíthatóság, illetve lekérdezhetőség kedvéért van szükség, viszont a könnyebb áttekinthetőség kedvéért ezeket elég csak akkor generálni, amikor már készen vagyunk a saját metódusok megírásával.

Változók: `kx,ky` a kör középpontja, `sugar` a kör sugara, `szog` a mutató vízszintessel bezárt szöge, `novekmeny` az az érték, amellyel a mutató majd arrébb megy, az `ido` pedig azt adja meg, hogy egy-egy lépés között mennyit várakozzon (aludjon). Az `aktiv` nevű változó segítségével tudjuk beállítani, hogy éljen-e a szál.

Még egy mozzanatot meg kell említeni: az óra majd működni fog, de nyilván nem az a fő cél, hogy az osztály ne unatkozzon, és jól elbíbelődjön magában, hanem az, hogy erről a tevékenységről értesüljön a nagyvilág. Erről közvetlenül csak a vezérlőt tudja értesíteni, Ehhez az `Ora` osztálynak ismernie kell a `Vezerlo` osztályt (vagyis kell, hogy legyen ilyen adattagja, amelynek értékét most a konstruktorban adjuk meg).

```
public class Ora extends Thread{

    private int kx, ky;
    private int sugar;
    private Color keretSzin;
    private Color mutatoSzin;
    private double szog;
    private boolean aktiv;
    private long ido;
    private double novekmeny;
    private Vezerlo vezerlo;

    public Ora(int kx, int ky, int sugar, Color keretSzin, Color mutatoSzin,
               double szog, boolean aktiv, long ido, double novekmeny,
               Vezerlo vezerlo) {
        this.kx = kx;
        this.ky = ky;
        this.sugar = sugar;
        this.keretSzin = keretSzin;
        this.mutatoSzin = mutatoSzin;
        this.szog = szog;
        this.aktiv = aktiv;
        this.ido = ido;
        this.novekmeny = novekmeny;
        this.vezerlo = vezerlo;
    }

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

Megjegyzés: A képlet nem pont az, ahogy órán vettük. Ha ugyanis figyelembe vesszük, hogy az origó a panel bal felső sarka, és lefelé van a pozitív irány, akkor itt függőlegesen ez a helyes képlet. Ha viszont következetesen a matematikában tanult irányokban gondolkozunk, vagyis a szögeket is az ott tanult irányítottságúnak tekintjük, akkor az órán vett módon is helyes értékeket kapunk.

Mint az elején már említettem, a mutató mozgását is meg kell oldanunk, ezért definiáltuk az osztályt a `Thread` osztály leszármazottjaként. Ahhoz viszont, hogy működjön is, meg kell írnunk a szál „lelkét”, a `run()` metódust. Ez felel majd azért, hogy mozogjon a mutató.

A mozgás egyszerűen oldható meg: mivel a mutatót a kör középpontja és a mutató vízszintessel bezárt szöge alapján tudjuk kirajzolni, ezért elég, ha ez a szög változik, mégpedig annak megfelelő értékkel, hogy mekkora ugrásokkal akarjuk bejárni a kört (ez lesz a `novekmeny`).

Vagyis, amíg fut a szál, addig növeljük a szöget, értesítjük erről a vezérlőt, majd altatjuk egy ideig a szálat. (Az, hogy altatás előtt vagy után növeljük, azon múlik, hogy induláskor akarjuk-e egy kicsit az eredeti állásban látni a mutatót, vagy sem.)

A metóduson belül tehát értesítjük a vezérlőt a változásról, aztán majd a vezérlő eldönti, hogy mit kezd evvel az információval. Dönthet úgy, hogy megkéri a panelt, hogy frissítse magát, de dönthetne úgy is, hogy pl. kiírja konzolra ezt a változást, vagy bármi mást csinál vele. Ha nagyon szigorúan vennénk a függetlenség elvét, akkor az `Ora` osztályban valahogy így kellene értesíteni a vezérlőt: `vezerlo.megvaltoztatottSzogErtek(szog)`, és teljes mértékben a `vezerlo` példányra bízni, hogy mit kezd evvel az információval, de talán a könnyebb érthetőség kedvéért megengedhetünk annyi slendriánságot, hogy csak frissítésre kérjük a vezérlőt. Ha kedve van hozzá, nyugodtan kipróbálhatja a szigorúbb változatot, viszont a vezérlő megfelelő metódusa ugyanaz lesz, mint most, de ekkor teljes mértékben rá bízza a döntést, míg most, a frissítés hívásával az `Ora` osztály látszólag kicsit leszűkíti a vezérlő mozgásterét.

```
@Override
public void run(){
    while(aktiv){
        try {
            szog += novekmeny;
            vezerlo.frissit();
            Thread.sleep(ido);
        } catch (InterruptedException ex) {
            Logger.getLogger(Ora.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

A rajzlást az `OraPanel` `paintComponent()` metódusa végzi (az `Ora` osztály `rajzolas()` metódusa csak azt mondja meg, hogy hogyan rajzoljon). A panel azonban nem is tud az Ora osztály létezéséről, nem is kell tudnia róla. Ő egy szerencsétlen „rabszolga”, aki kizárólag a vezérlővel tart kapcsolatot, és rajzoláskor szigorúan betartja a vezérlő rajzolásra vonatkozó utasításait. Vagyis azt csinálja, amit a vezérlő rajzolásra vonatkozó metódusa mond neki. Ehhez persze neki is ismernie kell a vezérlőt, azaz itt is kell ilyen adattag, és itt is biztosítani kell azt, hogy ez a mező értéket kaphasson. Ha továbbra is azt szeretnénk, hogy a panelt könnyen rá tudjuk húzni a frame-re, akkor paraméter nélküli konstruktorral kell rendelkeznie (azaz Java Bean-nek kell lennie), ezért most csak setterrel tudunk értéket adni a vezérlő példánynak Ezt nem tüntetem fel a kódban, nyilván egyedül is meg tudja csinálni.

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

    private Vezerlo vezerlo;

    public OraPanel() {
        initComponents();
    }

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

Megjegyzés: A `null` vizsgálat nagyon fontos (egyébként más kódrészletekben is), mert a panel `paintComponent()` metódusa már akkor megpróbál futni, amikor létrejön a panel, de még nincs is vezérlő példány.
Az órát valahol létre kell hozni, és el kell indítani a mozgását, vagyis el kell indítani a szálat. Ezt több helyen is megtehetjük.

A feladat a) pontja szerint a program indulásakor azonnal elindul az óra is, vagyis mihelyt létrejön a panel, azonnal meg kell hívnunk ezt az indító metódust is. Ezért vagy az `OraFrame` osztályban definiált, és a `main()` metódusból meghívott indító metódusban oldjuk meg, vagy a panel láthatóvá válásakor bekövetkező esemény hatására (ez utóbbi nem csak azért jobb, mert kényelmesebb, hanem azért is, mert így a frame független a paneltől, vagyis egyáltalán semmit sem kell tudnia, annak működéséről).

Mint már megbeszéltük, a panel semmit sem tud az `Ora` osztályról, ő kizárólag csak a vezérlővel van kapcsolatban, vagyis vele kell közölnie, hogy megjelent a felülete. Ismét írhatnánk úgy, hogy `vezerlo.megjelentaFelulet()`, de talán most is olvashatóbb a kód, ha utalunk rá, hogy milyen tevékenységet várunk a vezérlőtől.

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

Már meglehetősen sokszor hivatkoztunk a vezérlőre, épp ideje megbeszélni, hogy hogyan is épül fel. A vezérlő osztály mondja meg a panelnek, hogy mit rajzoljon, de önmagában emiatt nem feltétlenül kellene ismernie a panelt, hiszen ő csak „kiadja” a munkát, elég, ha a panel tudja, hogy kitől kell várnia azt. Azonban most a vezérlőnek is ismernie kell a panelt, mert ha a panel közepén akarja megjeleníteni az óra rajzát, akkor ismernie kell a panel méreteit, illetve majd őt fogja megkérni a rajz frissítésére is. Ezért a vezérlő osztályban kell majd egy, az órapanelre hivatkozó mező. Most az a legegyszerűbb, ha ennek a konstruktorban adunk értéket.
A rajzolás metódus egyszerű: meg kell mondania, hogy az óra `rajzolas()` metódusában leírt utasításokat kell alkalmazni.

Az eddigieket tartalmazó kódrészlet:

```
public class Vezerlo {

    private Ora ora;
    private OraPanel oraPanel;
    private final int SUGAR = 200;
    private final Color KERET_SZIN = Color.BLUE;
    private final Color MUTATO_SZIN = Color.RED;
    private final double SZOG = Math.PI/2;
    private final boolean AKTIV = true;
    private final long IDO = 1000;
    private final double NOVEKMENY = -Math.PI/6;

    public Vezerlo(OraPanel oraPanel) {
        this.oraPanel = oraPanel;
    }

    public void rajzolas(Graphics g) {
        if(ora != null) ora.rajzolas(g);
    }
```

```
public void oraInditas() {
    int kx = oraPanel.getWidth()/2;
    int ky = oraPanel.getHeight()/2;
    ora = new Ora(kx, ky, SUGAR, KERET_SZIN, MUTATO_SZIN, SZOG,
                          AKTIV, IDO, NOVEKMENY,this);
    frissit();
    ora.start();
}

public void frissit() {
    oraPanel.repaint();
}
```

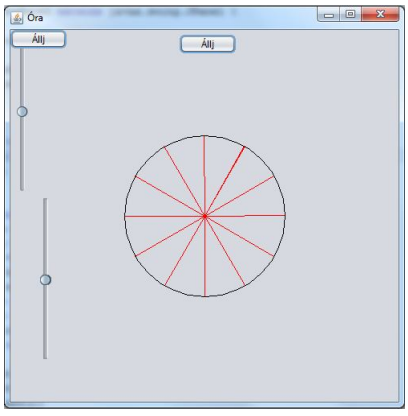
Megjegyzések:

1. Ha csak kirajzolni akarjuk az órát (és nem a szálat indítani), akkor az `ora.start();` hívás helyett előfordulhat, hogy szükség van frissítésre. Érdekes módon néha enélkül is látjuk a rajzot, de ténylegesen kellene.

2. A másik **fontos** észrevétel: a felületek összeállításakor a NetBeans már a panel felrakásakor, azaz tervezési nézetben is „kipróbálja" a panel metódusait (ha például olyankor húzzuk rá a panelt a frame felületére, amikor már meg van írva a szál `run()` metódusa, akkor már tervezési nézetben is mozogni látjuk a mutatót). Egy próbát megér, de a továbbiakban is tartsa magát ahhoz az elvhez, hogy még a kódírás előtt rakja fel a panelt.
Ugyancsak futtatni akarja a `paintComponent()` metódust is. Ekkor azonban (vagyis a tervezési fázisban) még nem is létezik a vezérlő objektum. Emiatt, ha a null érték vizsgálata nélkül most akarjuk ráhúzni a panelt a frame-re, `NullPointerException` hibaüzenetet kapunk. Ezt a hibaüzenetet akkor is megkaphatjuk, ha még a metódus megírása előtt felraktuk a panelt. Bár ennek ellenére működik a program, de mégsem tökéletes. Ha el akarja kerülni ezt a hibaüzenetet, (márpedig akarja! ☺), akkor a `paintComponent()`-ben mindenképpen kell a null-vizsgálat, és hasonlóan a `Vezerlo` osztály `rajzolas()` metódusában is.

3. Kicsit félve írom ezt a megjegyzést, mert ha nem említeném, lehet, hogy eszébe sem jutna kipróbálni, de mégis leírom, nehogy az legyen, hogy örül egy jónak tűnő megoldásnak, holott nem jó.
Esetleg feltűnhet, hogy a frissítés metódus „kiváltható" avval, ha a `paintComponent()` metóduson belül hívja meg a `repaint()` utasítást. Ez **hibás** ötlet, hiszen épp most írt meg egy durva végtelen ciklust, ugyanis a `repaint()` a `paintComponent()` metódust hívja meg. Hogy akkor miért működik? Mert a futtatókörnyezet optimalizál. De akkor sem szabad erre támaszkodni, SOHA ne írjon végtelen ciklust!

4. Utolsó megjegyzés: Egy animációt elvileg úgy kellene megírni, hogy egy ciklusban töröljük a régi rajzot, majd megrajzoljuk az újat. De most vajon ki törli a régit? Próbálja ki, hogy kikommentezi a `paintComponent()` metódus `super.paintComponent()` hivatkozását. Láthatja, hogy ekkor megmaradnak a régi rajzok is. (Sőt, ha olyankor próbálja ki, amikor már a gomb és a csúszka is rajta van, akkor még érdekesebb élményben lesz része. ☺) Vagyis a `super.paintComponent()` az, aki állandóan frissíti az őst, vagyis az eredeti `JPanel`-t.



b/ feladat: A felületre kattintva változzon meg az óramutató iránya:

`OraPanel`:

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    vezerlo.iranyValtas();
}
```

Természetesen a `mouseClicked` esemény is jó. A `mousePressed` az egér megnyomásakor, a clicked az egérgomb felengedésekor „tüzel”.

`Vezerlo`:

```
public void iranyValtas() {
    ora.iranyValtas();
}
```

`Ora`:

```
void iranyValtas() {
    novekmeny = -novekmeny;
}
```

c/ feladat: Rakjunk fel az `OraPanel`-re egy inditó gombot, és definiáljuk a hozzá tartozó eseményt, de mi történjen a gombnyomás hatására?

Ha az volna a feladat, hogy a gombnyomás hatására jelenjen meg és induljon el az óra, akkor a felület betöltése helyett itt kellene meghívnunk vezérlő `oraInditas()` metódusát. A feladat azonban az, hogy az óra rajza a program indításakor azonnal jelenjen meg, de csak a gombnyomás hatására induljon el. Ez azt jelenti, hogy az óra létrehozását és a szál elindítását külön kell választanunk, vagyis nem írhatjuk ugyanabba a metódusba őket. Folytatva a gondolatmenetet, ha az lenne a feladat, hogy azonnal lássuk az óra rajzát, de csak gombnyomásra induljon, és járjon „örökké”, akkor elég lenne annyi módosítás, hogy a vezérlő `oraInditas()` metódusából kivesszük az `ora.start()` metódushívást, és átrakjuk abba, amelyet a gombnyomás eseményekor hívunk meg. A feladat azonban ennél kicsit összetettebb. Egyrészt **egy szálat csak egyszer lehet elindítani**, másrészt a gombnyomáshoz több funkció is tartozik: ha nem megy, induljon el, ha megy, álljon meg. Ezért itt nem elég a `start()`-ot indítani, hanem működésváltásra kell kényszerítenünk az órát. De persze, erről a panel mit sem tud, neki elég csak annyit tudnia, hogy a gombnyomás hatására szólnia kell a vezérlőnek. Ismét lehetne annyi, hogy `vezerlo.megnyomtakaGombot()`, de talán olvashatóbb a kód, ha ismét utalunk rá, hogy mit várunk a vezérlőtől.

Egyúttal a gomb feliratváltását is beszéljük meg. Legegyszerűbb, ha azt vizsgáljuk, hogy épp mi a felirat, és az ellenkezőre váltunk. De persze, nem kellene beégetni, hanem pl. az osztály elején, konstansként lehetne megadni (vagy egy `Global` osztályban):

```
private final String EREDETI = "Indulj";
private final String UJ = "Állj";
```

```
    private void btnInditoActionPerformed(java.awt.event.ActionEvent evt) {
        vezerlo.mukodesValtas();
        if(btnIndito.getText().equals(EREDETI)){
            btnIndito.setText(UJ);
        }else{
            btnIndito.setText(EREDETI);
        }
    }
```

A gomb felrakásához egy kis kitérő:
Akinek kedve van hozzá, és kipróbálja, hogy csak most, ennél a feladatnál rakja fel a gombot, az eljátszadozhat a layout-okkal is. Ha az `OraPanel` layout-ját pl. meghagyjuk free design-nak, vagy átállítjuk flow-, border-, stb. layout-ra, akkor futtatáskor nem látszódik a gomb. Le kell szedni a frame-ről a korábbi panelt, és fordítás után ismét ráhúzni. Ez ugyanavval a macerával jár, mint amit már korábban is említettem, ráadásul most a frame-n ki kellene kommentezni az `oraPanel1`-re való hivatkozást (és a panel felrakása után kiszedni a kommentet), hiszen leszedtük a panelt. Különben nem tudjuk lefordítani az osztályokat, így nem tudjuk ismét felhúzni.

Ha viszont `Null Layout`-ra állítjuk a panelt, akkor nem kell újra-generálni, rendesen fog működni, bár a gomb méretét lehet, hogy kicsit nagyobbra kell állítani. Ízlés kérdése, de én sokszor inkább a `Null Layout`-ot használom, mert bár talán kicsit több mindent kell beállítani, de semmilyen layout-manager nem erőszakoskodik az akaratom felett.

Az lenne szép, ha a gomb vízszintesen mindig középen lenne. Evvel is eljátszadozhat, ha kedve tartja: Ha flow layout-ot használ, akkor a lap tetején, vízszintesen középen lesz, de ekkor hova kerül majd később a csúszka? Vagyis ez hosszabb távon nem jó megoldás. Lehet az, hogy két panelen dolgozunk, a felső egy keskeny panel, csak a gombot tartalmazza, ekkor ez lehet flow layout. A másik lehetőség, hogy marad a `Null Layout`, és a `setBounds()` metódussal beállítjuk a gomb helyét – természetesen a panel szélességének függvényében. Ekkor viszont az a baj, hogy átméretezéskor marad az eredeti helyen, vagyis elkerül középről. Ennek megoldása, hogy figyeljük az átméretezés eseményt (`componentResized`), és az esemény bekövetkeztekor a gomb határait ismét átállítjuk az épp aktuális szélességnek megfelelően.
Egy másik „megoldás”: a „lusta ember” „praktikus” megoldása: letiltjuk a panel átméretezhetőségét. ☺) Az viszont **fontos** javaslat, hogy ezt csak akkor tegye, amikor már működik a program, mert a fejlesztés közben sokat segíthet, ha a próbálkozások során tudja változtatni a felületet.

Térjünk vissza a kódokhoz. A gombnyomás hatására akarjuk elindítani az órát, de csak akkor, ha még nem jár. Ha már jár, akkor viszont működésváltásra akarjuk kérni. Így tehát az óraindítást ahhoz a feltételhez kötjük, hogy még nem él az óra-szál. A `Vezerlo` osztály megfelelő metódusa:

```
    public void mukodesValtas() {
        if (!ora.isAlive()) {
            ora.start();
        } else {
            ora.mukodesValtas();
        }
    }
```

Figyeljen rá, hogy az `oraInditas()` metódusból szedje ki a szálindítást (`ora.start()`), ott csak az óra definiálása marad, illetve beírhat helyette egy `this.repaint()` hívást. Mivel a gombnyomáskor feltételhez kötjük az óra indulását, most azon kívül, hogy nem a feladatkiírás szerint működik, nem lenne baj az ottfelejtett szálindítással, de arra mindig nagyon figyeljen, hogy egy szálat tilos egynél többször elindítani. (Nem is lehet, mert elszáll tőle a program.)

A működésváltáshoz szükségünk van egy logikai változóra (az `Ora` osztályban), hiszen ez mutatja, hogy az óra épp működik-e vagy sem. Legyen ez a boolean `mukodhet` változó.
A gombnyomás hatására – vagyis a `mukodesValtas()` metódus meghívásakor ennek a változónak az értéke ellenkezőjére változik. Ha ez az érték igaz, akkor az óramutatónak mozognia kell, egyébként állnia.

Ha nem működhet, akkor várakoztatnunk kell a szálat. Mivel nem tudjuk, hogy meddig kell állnia, ezért a `wait()` metódust kell használnunk. Várakozni azonban csak a szál tud, vagyis csak a `run()` metódusban lehet kiadni ezt a parancsot. Természetesen úgy is meg lehet oldani, hogy nem közvetlenül a `run()` metódusban hívjuk meg, hanem egy másik metódusban, amelyet a `run()` aktivizál. Most ezt a megoldást választjuk.

Ha eddig állt, és most engedjük működni, akkor viszont csak fel kell engednünk a várakozó szálat, vagyis a `notify()` hivatkozást kell alkalmaznunk.

Még egy fontos dologra oda kell figyelni: a várakozásnak és az ébresztésnek szinkronban kell lennie egymással, ezért szinkronizálni kell az őket tartalmazó metódusokat (vagy blokkokat).

De ki dönti el, hogy az óra működhet-e vagy sem? Természetesen az, aki megnyomja az indító gombot. Ennek hatására – ahogy korábban már megbeszéltük – a panel jelez a vezérlőnek, az pedig megkéri az óra példányt, hogy változtassa meg a működését, azaz meghívja az `ora` példány `mukodesValtas()` metódusát. Ez a metódus a `mukodhet` változó értékét ellenkezőjére állítja. Az `Ora` osztály metódusa:

```
private boolean mukodhet;

public synchronized void mukodesValtas() {
    mukodhet = !mukodhet;
    if(mukodhet) notify();
}
```

Ha nem működhet, akkor a várakozásról a `run()` metódus gondoskodik.

Szerintem egyszerűbb, ha a várakoztatást egy külön kis metódusban oldjuk meg, mégpedig azért, mert talán könnyebben érthető, ha ezt a külön metódust szinkronizálunk, és nem egy metódus belsejének egyetlen blokkját. De majd ez utóbbi megoldást is megmutatom.

A várakozással bővített `run()` metódus (most külső metódusban hívjuk meg a `wait()`-et):

```
@Override
public void run(){
    while(aktiv){
        try {
            varakozik();
             szog += novekmeny;
             vezerlo.frissit();
             Thread.sleep(ido);
        } catch (InterruptedException ex) {
            Logger.getLogger(Ora.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

private synchronized void varakozik() throws InterruptedException {
    if(!mukodhet){
        wait();
    }
}
```

Az ígért másik változat:
Természetesen nem feltétlenül muszáj külön metódus a várakozásra (bár talán érhetőbb és áttekinthetőbb így), ezt a néhány sort közvetlenül is beírhatjuk a `run()` metódusba. Viszont nem az egész metódust akarjuk szinkronizálni (nincs mivel szinkronizálni), hanem csak ezt a részletet, azaz csak ezt a blokkot kell szinkronizálnunk:

```
while (fut) {
    if(!mukodhet){
        synchronized (this) {
            try {
                this.wait();
            } catch (InterruptedException ex) {
                Logger.getLogger(Ora.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Örülhetünk, hogy készen vannak a szükséges metódusok, de sajnos az óra mégsem indul. (Viszont ha többször is megnyomja a gombot, akkor láthatja, hogy a működésváltás helyesen viselkedik.) Azért nem indul, mert alapból `false` a `mukodhet` változó értéke, és csak a második gombnyomás hatására változik meg, hiszen csak ekkor hívjuk meg az óra `mukodesValtas()` metódusát. Az első gombnyomás hatására indul el az óra szál. Ezért be kell állítanunk, hogy induláskor `true` legyen a `mukodhet` változó értéke.

A működésváltással kapcsolatban még egy dologra oda kell figyelnünk: nyilván nem mindegy, hogy mi a gomb felirata. Vagyis működésváltáskor ezt is változtatnunk kell. Ez a panel feladata, hiszen rajta van a gomb, ezt oldottuk meg a gombnyomás metódusban.

d/ feladat: Tegyünk fel a panelre egy csúszkát (`slider`). Ennek orientációja (`orientation`) legyen `VERTICAL`, és legyen átlátszó (`opaque` kikapcsolva).

Állítsuk be a minimális, maximális értékét (`minimum`, `maximum`). Ezeket beállíthatjuk a Properties-ben is, de aki jobban szereti, használhatja a `set()` metódusokat. (Minimumként nullánál nagyobb értéket állítson be, ez lesz ugyanis a szálban a minimális várakozási idő.)

Az `OraPanel` megfelelő metódusa pl.:

```
private void csuszkaStateChanged(javax.swing.event.ChangeEvent evt) {
    vezerlo.idoValtozas(csuszka.getValue());
}
```

A `Vezerlo` osztályban:

```
public void idoValtozas(int ujIdo) {
    if(ora != null) ora.setIdo(ujIdo);
}
```