Lepkevadászat

Az eddigiek összefoglalásaként írjunk egy kis lepkefogó játékot!

Az 1000×600-as belső felület három részre oszlik. A jobboldalán egy 200×600-as felületen lehet adminisztrálni a játékosokat, illetve az itt lévő indító gomb hatására elindítani a játékot. A játéktér 800×550 pixel méretű, a felső, 50 pixel magas rész pedig arra szolgál, hogy itt olvasható majd a hátralévő játék-idő és az elért pontszám is.

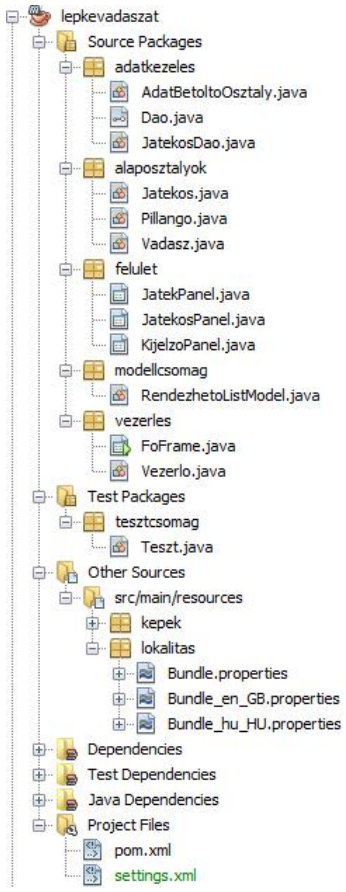
Induláskor adjuk meg a játékos nevét, majd az indító gomb hatására kezdődjön a játék. A játék során a mező fölött időnként megjelenik egy-egy lepke, és elkezd repülni – most elég egyszerűen, csak négy lehetséges irányban mozog egyenletes, de véletlen sebességgel. (Véletlen pozíció, véletlen méret, véletlen sebesség, véletlen, hogy milyen irányban indulnak: balra le/föl vagy jobbra le/föl ↙↖↘↗). A gomb megnyomásakor a lepkevadász is megjelenik a mező jobb alsó sarkában. Őt az egérrel tudjuk mozgatni, és az a cél, hogy minél több lepkét el tudjon kapni. Az elkapott lepkékért egy-egy pontszám jár.

Ha lejárt a játékideje, a listában jelenjen meg a játékos neve és a pontszáma. Ha egy játékos többször is játszik, akkor csak a legutolsó eredménye jelenjen meg (vagy ha lágyabb szívű, akkor a legjobb). A listafelületen pontszám szerint csökkenően rendezve legyenek az adatok.

A Mentés gomb hatására a listában lévő adatokat mentsük el egy adatbázisba, a következő induláskor az innen olvassuk ki a játékosok adatait, és jelenítsük is meg a listafelületen.

Mentéskor értelemszerűen csak az újakat kell beszúrni, a korábban is létezőket pedig módosítani. Ehhez beágyazott Derby-t használunk majd Mavennel.

Néhány megoldásrészlet:



A feladat elég komplex ahhoz, hogy segítségével az eddigiek nagy részét átismételhetjük.

Bár most a játék részével kezdjük, de mivel majd adatbázis-kezelés is lesz benne, ezért célszerű Maven projektet készíteni. A játék sikerében is reménykedünk, ezért eleve arra készítjük fel, hogy több nyelven is megszólalhasson, de az ide vonatkozó kód-részletekről csak a megoldás végén lesz szó.

És persze, a tesztelésről sem feledkezünk meg, amire a *pom.xml* fájlban is fel kell készülnünk. A *pom.xml* fájl letölthető a képeket tartalmazó *lepkevadaszat.zip* adatfájlból.

A projekt szerkezete látható a jobboldali ábrán.

A feladatot egyetlen panellel is meg lehetne oldani, de három részre szedve jobban elkülönülnek a funkciók, illetve könnyebben lehet módosítani további igények esetén, hiszen bármelyik három bármikor helyettesíthető egy másik panellel.

A `JatekPanel`-en zajlik a játék. Ha csak ezt az egyetlen panelt tennénk fel a frame-re, akkor is lehetne játszani, csak persze, jóval kevesebbet tudna a játék. Betöltéskor azonnal elindulnának a pillangók, és a vadász tudná fogdosni őket. (Annyit kellene csak változtatni a kész projekthez képest, hogy az indítást nem a gombnyomás, hanem a form betöltésének eseményéhez rendelnénk.)

Ha ehhez hozzárakjuk a `KijelzoPanel`-t, akkor már az idő múlását is láthatnánk, a pontokat is, és akkor már akár innen is indíthatnánk a játékot (ha kiegészítjük a felületet egy gombbal). Ez a két panel is tud együtt dolgozni, illetve akkor is működőképes a projekt, ha csak ez a két panel szerepel a frame-n.

Ha még a `JatekosPanel`-t is hozzárakjuk, akkor játékosokat is tudunk rendelni a játékhoz, és az eredményüket is tudjuk kezelni. Elvileg akkor is működőképes a projekt (ugyancsak egy minimális módosítás árán), ha csak ez és a `JatekPanel` van fent a frame-n, de így kevésbé lenne érdekes a játék.

Ahogy már megszokta, és remélhetőleg nem csak megszokta, hanem érti is és egyet is ért evvel a logikával, a panelek közti kapcsolatot, illetve a pillangókat, vadászt, játékosokat is a `Vezerlo` kezeli.

Az adatkezelést majd később részletezzük, most csak annyit jegyzünk meg, hogy az úgy-nevezett Dao (data access object) tervezési minta konvencióinak megfelelően alakítottuk ki az alapot jelentő `Dao` nevű interfészt, és ezt implementálja a `JatekosDao` osztály. Mint már egy korábbi példán is láttuk, célszerű az adatbevitelt külön szálon futtatni, hogy ne tartsa fel a kirajzolást. Ezt oldja meg az `AdatBetoltoOsztaly`. A továbbiakban néhány részlet:

Alaposztályok:

A `Jatekos` osztályból csak néhány kis kódrészletet emelünk ki:

Mivel rendezhető listamodellbe szeretnénk rakni a játékos példányokat, ezért összehasonlíthatóknak kell lenniük:

```
public class Jatekos implements Comparable<Jatekos>{

    @Override
    public int compareTo(Jatekos o) {
        return o.pontSzam - this.pontSzam;
    }
```

Mivel egy játékos a játékból is és az adatbázisból is létrehozható, ezért két konstruktort is írunk, illetve közöljük még a `pontotKap()` metódus kódját is:

```
    //Akkor hívjuk meg, amikor a játékban létrehozunk egy példányt
    public Jatekos(String nev) {
        this.nev = nev;
    }

    // Az adatbázisból való betöltéskor példányosítunk vele.
    public Jatekos(String nev, int pontSzam) {
        this.nev = nev;
        this.pontSzam = pontSzam;
    }

    public void pontotKap(){
        pontSzam++;
    }
```

A pillangó mozog, ezért szálként oldjuk meg:

```
public class Pillango extends Thread{

    private Image kep;
    private int kepX, kepY;
    private int kepSzelesseg, kepMagassag;
    private Vezerlo vezerlo;
    private long ido;
    private int dx;
    private int dy;
    private boolean elkaptak;

    private static int feluletSzelesseg, feluletMagassag;
```

```java
public Pillango(Image kep, int kepX, int kepY,
                int kepSzelesseg, int kepMagassag,
                int dx, int dy, Vezerlo vezerlo, long ido) {
    this.kep = kep;
    this.kepX = kepX;
    this.kepY = kepY;
    this.kepSzelesseg = kepSzelesseg;
    this.kepMagassag = kepMagassag;
    this.dx = dx;
    this.dy = dy;
    this.vezerlo = vezerlo;
    this.ido = ido;
}

public void rajzolas(Graphics g){
    g.drawImage(kep, kepX, kepY, kepSzelesseg, kepMagassag, null);
}

@Override
public void run() {
    // Addig mozog, amíg a felület felett van és nem kapták el.
    while (kepX > -kepSzelesseg && kepX < feluletSzelesseg &&
           kepY > -kepMagassag && kepY < feluletMagassag && !elkaptak) {
        mozdul();
        frissit();
        kesleltet();
    }
    if(elkaptak) vezerlo.torol(this);
}

// A kezdeti irányt tartva ferdén mozog.
private void mozdul() {
    kepX += dx;
    kepY += dy;
}

private void frissit() {
    vezerlo.frissit();
}

private void kesleltet() {
    try {
        Thread.sleep(ido);
    } catch (InterruptedException ex) {
        Logger.getLogger(Pillango.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
    /**
     * Akkor fogták meg, ha az adott koordinátájú pont a pillangó
     * képén belülre esik.
     *
     * @param kx
     * @param ky
     */
    public void megfogtak(int x, int y) {
        elkaptak = (kepX < x && x < kepX + kepSzelesseg &&
                    kepY < y && y < kepY + kepMagassag);
    }
```

+ setterek, getterek.

A vadászt csak ki kell rajzolni, illetve ellenőrizni, hogy valóban megfogtuk-e az egérrel:

```
public class Vadasz {
    private Image kep =
     new ImageIcon(this.getClass().getResource("/kepek/vadasz.gif")).getImage();

    private int kepX, kepY;
    private static int kepSzelesseg;
    private static int kepMagassag;

    public Vadasz(int kepX, int kepY) {
        this.kepX = kepX;
        this.kepY = kepY;
    }

    public void rajzolas(Graphics g){
        g.drawImage(kep, kepX-kepSzelesseg/2, kepY-kepMagassag/2,
                                kepSzelesseg, kepMagassag, null);
    }

    public boolean megfogtak(int x, int y) {
        return kepX - kepSzelesseg/2 <= x && x <= kepX + kepSzelesseg/2 &&
               kepY-kepMagassag/2 <= y && y <= kepY + kepMagassag/2;
    }
```

+ setterek, getterek.

Két panel:

A `JatekPanel` és a `KijelzoPanel` feladata nagyon egyszerű: végre kell hajtaniuk a vezérlő parancsait (na jó, teljesíteniük kell a vezérlő kéréseit), illetve jelezni a vezérlőnek, ha valamilyen esemény történt rajtuk.

A harmadik panel feladatáról majd kicsit később esik szó.

A `JatekPanel` két metódusa (a deklarációk, setterek és generált részek nélkül):

```
// Kirajzolja a háttérképet, és azt, amit a vezérlő mond neki.
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    int kezdox = 0, kezdoy = 0,
        szelesseg = this.getWidth(),
        magassag = this.getHeight();
    g.drawImage(hatterKep, kezdox, kezdoy, szelesseg, magassag, null);

    if(vezerlo != null) vezerlo.rajzol(g);
}
```

```
// Szól a vezérlőnek, hogy arrébb mozgatták az egeret.
private void formMouseDragged(java.awt.event.MouseEvent evt) {
    vezerlo.arrebbMozgat(evt.getX(), evt.getY());
}
```

Esetleg próbálkozhat a `formMouseMoved` eseménnyel is, és figyelje meg, hogy melyikkel könnyebb játszani. ☺

A `KijelzoPanel` metódusai ugyancsak generált rész és setter nélkül:

```
private ResourceBundle bundle;

public void idotKiir(long aktIdo) {
    lblIdo.setText(String.format(bundle.getString("KijelzoPanel.lblIdo.text"),
                                                                    aktIdo));
}

public void pontszamotKiir(int pontSzam) {
    lblPontszam.setText(bundle.getString("KijelzoPanel.lblPontszam.text")
                                                            + pontSzam);
}

public void jatekVeg() {
    lblIdo.setText("");
    lblPontszam.setText("");
}
```

Következzen a **`Vezerlo`** osztály, de hogy ne kelljen több részletben közölni, ezért előbb beszéljünk még meg valamit, mégpedig azt, hogy hogyan keletkeznek a pillangók.

A feladat szerint bizonyos időközönként keletkeznek, ezért szükségünk van valamilyen időzítőre. Talán legegyszerűbb a Timer használata lenne, de most a szálkezelés gyakorlása a célunk, ezért ezt is szállal oldjuk meg (ami egyébként nem bonyolultabb a Timer használatánál.)

Kérdés, hogy honnan szedjünk szálat. Erre több megoldás is kínálkozik:

1. Hozzunk létre egy új szál osztályt, (pl. `PillangoGenerator extends Thread {...}`), és az indító gomb hatására ezt indítsuk el.
2. Nem kell új osztály, a vezérlő is viselkedhet szálként.
3. Mivel úgyis a kijelző panel mutatja az idő múlását, viselkedjen ő szálként.

Mindhárom megoldás elfogadható, bár a harmadik talán kicsit nehézkesebb, mint az első kettő. Vagyis nagyrészt szubjektív, hogy ki melyik megoldást választja.

Mint már volt szó róla, az adatbeolvasást is külön szálon oldjuk meg. Ha ezt a vezérlőre bíznánk, akkor nyilván már nem tudná kezelni a pillangókat. Most válasszuk azt a megoldást, hogy az adatbevitelt rakjuk külön osztályba, és a vezérlőre bízzuk a pillangók kezelését, vagyis erre nem írunk külön osztályt. (De persze, más jó megoldás is lehet.)

Szálat kétféle módon hozhatunk létre:

– kiterjesztjük a `Thread` osztályt;
– implementáljuk a `Runnable` interfészt, és az így kapott osztály egy példányát adjuk át egy `Thread` példánynak.

Még egy kérdés: hogyan lehet megoldani, hogy több játékot is lehessen indítani egymás után. Egy szálat csak egyszer lehet elindítani, ezért ebben is kétféle lehetőségünk van:

– Minden indításkor új szálat hozunk létre. (Ekkor persze gondoskodni kell arról is, hogy a korábbi szál leálljon, de ez sokszor a `run()` metódus ügyes megírásával is megoldható.)
– Egyetlen szállal dolgozunk, ha a játékos ideje lejárt, akkor várakoztatjuk a szálat, ha új játékos jelentkezik, akkor pedig felengedjük. Esetünkben ez most bonyolultabb megoldásnak tűnik.

Most a vezérlő osztályt a `Runnable` interfész implementálásaként definiáljuk, és minden játékindításkor új szálat hozunk létre. **FONTOS**: Új szál létrehozása jóval nehezebb lenne akkor, ha `Thread`-ként deklaráltuk volna a vezérlő osztályt. Így, hogy `Runnable` típusú, bármikor létrehozhatunk belőle egy-egy szál osztályt.

A `run()` metódus a játékidő lejártáig fut, utána leáll, ezért nem kell külön foglalkoznunk a szál leállításával. A pillangók `run()` metódusa is csak addig fut, ameddig a pillangó rajz-

felület fölött van vagy még nem kapták el, ezért itt is csak arra kell figyelni, hogy az elkapásakor kiszedjük a rajzolandók listájából. (Szálat lehetőleg a `run()` metódus ügyes megírásával szüntessünk meg, ne használja a `stop()` metódust, mert az elavult és nem biztonságos.)

Hogy ne kelljen később módosítani, most úgy közöljük a `Vezerlo` osztály kódját, mintha már készen lenne a `JatekosPanel` is. Ha ki akarja próbálni még az adatkezelés előtt (akarja, mert fontos, hogy lássa az eddigi munkája eredményét), akkor az ottani Indítás gomb megnyomásához egyelőre rendelje hozzá a vezérlő osztály `jatekInditas()` metódusát egy fitív `Jatekos` példány paraméterrel.

```
// Lehetne Thread is, de akkor nem tudnánk többször újra-indítani.
// Ebből viszont lehet többször is létrehozni egy-egy szálat.
public class Vezerlo implements Runnable{

    private final long JATEK_IDO_SEC = 15;
    private final int ALSO_IDO_MSEC = 10;
    private final int FELSO_IDO_MSEC = 50;
    private final int ALSO_PILLANGO_MERET_PIXEL = 50;
    private final int FELSO_PILLANGO_MERET_PIXEL = 100;

    private final long KELETKEZESI_IDO_MSEC = 1000;
    private final int VADASZ_SZELESSEG = 150;
    private final int VADASZ_MAGASSAG = 150;

    private JatekPanel jatekPanel;
    private KijelzoPanel kijelzoPanel;
    private JatekosPanel jatekosPanel;

    private List<Image> kepek = new ArrayList<>();
    private List<Pillango> pillangok = new CopyOnWriteArrayList<>();
    private Vadasz vadasz;
    private Jatekos jatekos;
    private List<Jatekos> beolvasottJatekosok = new ArrayList<>();

    public Vezerlo(JatekPanel jatekPanel, KijelzoPanel kijelzoPanel,
                   JatekosPanel jatekosPanel) {
        this.jatekPanel = jatekPanel;
        this.kijelzoPanel = kijelzoPanel;
        this.jatekosPanel = jatekosPanel;
    }

    public void beallitas() {
        // Beállítja a vadász méretét.
        Vadasz.setKepSzelesseg(VADASZ_SZELESSEG);
        Vadasz.setKepMagassag(VADASZ_MAGASSAG);

        // Beállítja a felület méretét.
        Pillango.setFeluletSzelesseg(jatekPanel.getWidth());
        Pillango.setFeluletMagassag(jatekPanel.getWidth());
    }
```

Mivel a játékpanel azt rajzolja, amit a vezérlő mond neki, ezért itt kell megadni a vadász kirajzolására vonatkozó leírást is, természetesen a pillangók rajzolást leíró metódusának meghívása mellett.

```
public void rajzol(Graphics g) {
    for (Pillango pillango : pillangok) {
        pillango.rajzol(g);
    }
    if(vadasz != null) vadasz.rajzol(g);
}

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

```
/**
 * Bizonyos időnként elindít egy-egy pillangót,
 * az idő lejárta után jelzi a paneleknek, hogy ez a játék véget ért
 * és elvégzi a leállításkor szükséges műveleteket.
 */
@Override
public void run() {
    long hatralevoIdo = JATEK_IDO_SEC * 1000;
    while (hatralevoIdo >= 0) {
            try {
                kijelzoPanel.idotKiir(hatralevoIdo / 1000);
                pillangotIndit();
                Thread.sleep(KELETKEZESI_IDO_MSEC);
                hatralevoIdo -= KELETKEZESI_IDO_MSEC;
            } catch (InterruptedException ex) {
                Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE,
                                    null, ex);
            }
    }
    kijelzoPanel.jatekVeg();
    jatekosPanel.jatekVeg(jatekos);
    leallit();
}
```

(Ha a `JatekosPanel` osztály megírása előtt akarja kipróbálni, akkor természetesen innen még kimaradhat a `jatekVeg()` hívás. Természetesen a frame-n még össze kell ismertetni egymással a paneleket és a vezérlőt.)

```java
/**
 * Beállítja a szükséges adatokat, létrehoz egy pillangót, és elindítja.
 */
private void pillangotIndit() {
    int kepSzelesseg = (int) (Math.random()
                    *(FELSO_PILLANGO_MERET_PIXEL-ALSO_PILLANGO_MERET_PIXEL)
                    + ALSO_PILLANGO_MERET_PIXEL);
    int kepMagassag = (int) (Math.random()
                    *(FELSO_PILLANGO_MERET_PIXEL-ALSO_PILLANGO_MERET_PIXEL)
                    + ALSO_PILLANGO_MERET_PIXEL);
    int kepX = (int) (Math.random()*(jatekPanel.getWidth() - kepSzelesseg));
    int kepY = (int) (Math.random()*(jatekPanel.getHeight() - kepMagassag));
    long ido = (long) (Math.random()*(FELSO_IDO_MSEC - ALSO_IDO_MSEC)
                    + ALSO_IDO_MSEC);
    int kepIndex = (int) (Math.random()*kepek.size());
    Image kep = kepek.get(kepIndex);
    int dx = (Math.random() < 0.5) ? 1 : -1;
    int dy = (Math.random() < 0.5) ? 1 : -1;

    Pillango pillango = new Pillango(kep, kepX, kepY,
                                     kepSzelesseg, kepMagassag,
                                     ido, dx, dy, this);
    pillangok.add(pillango);
    pillango.start();
}
```

```java
/**
 * Elindítja az aktuális játékot.
 */
public void jatekInditas(Jatekos jatekos) {

    this.jatekos = jatekos;

    vadasz = new Vadasz(jatekPanel.getWidth()-Vadasz.getKepSzelesseg()/2,
                        jatekPanel.getHeight()-Vadasz.getKepMagassag()/2);
    frissit();

    // Létrehoz és elindít egy új szálat,
    // ez a szál hozza létre a pillangókat
    Thread szal = new Thread(this);
    szal.start();
}
```

Ha a pillangót elkapták, akkor ezt a törlő metódust hívja meg, a játékidő lejártakor pedig a játékot leállító metódust.

```
public void torol(Pillango pillango) {
    pillangok.remove(pillango);
    jatekos.pontotKap();
    kijelzoPanel.pontszamotKiir(jatekos.getPontSzam());
}

private void leallit() {
    pillangok.clear();
    vadasz = null;
    frissit();
}
```

Még azt kell megbeszélnünk, hogy egyáltalán hogyan mozog a vadász, és hogy kapja el a pillangókat. Az egérmozdítás (dragged, azaz a gombot nyomva való mozgatás) tényét a játékpanel már közölte a vezérlővel, most csak a vezérlő reagálását kell megírnunk:

```
/**
 * Ha létezik, és a játékfelület fölött van a vadász, akkor az
 * adott pozícióra állítjuk a középpontját, és megvizsgáljuk, hogy
 * fogott-e lepkét.
 *
 * @param x
 * @param y
 */
public void vadasztArrebbMozgat(int x, int y) {
    if(vadasz != null && vadasz.megfogtak(x,y)){

        vadasz.setKepX(x);
        vadasz.setKepY(y);

        // Ez egy kis beégetés, a kép alapján nagyjából itt van
        // a lepkeháló középpontja.
        int haloKozepX = x - Vadasz.getKepSzelesseg() / 4;
        int haloKozepY = y - Vadasz.getKepMagassag() / 4;
        // fogott-e lepkét
        lepkeFogas(haloKozepX, haloKozepY);
    }
}

private void lepkeFogas(int x, int y) {
    for (Pillango pillango : pillangok) {
        pillango.megfogtak(x,y);
    }
}
```

Ha megadjuk a játékos nevét, akkor már lehet is játszani, viszont a játékosok egy része hiú, és szeretné elmenteni az eredményét, ezt oldja meg a játékos panel.

Ha csak az aktuális játék eredményeit akarjuk kijelezni, az is nagyon egyszerű, hiszen csak rá kell írni a listafelületre a játékosok nevét. Rendezetten szeretnénk, ezért rendezhető lista-modellbe rakjuk őket. (Vagy írja meg önállóan, vagy letöltheti az adatokat tartalmazó fájlból.)

Most úgy oldjuk meg, hogy a listában mindenki csak egyszer szerepelhet, és a legfrissebb eredményét lehet olvasni. Módosítsa úgy, hogy mindig a legjobb eredmény látszódjon.

A `JatekosPanel` osztály indító és eredményíró metódusai (setter, getter nélkül):

```
private RendezhetoListModel<Jatekos> jatekosModell =
        new RendezhetoListModel<Jatekos>();

public JatekosPanel() {
    initComponents();
    btnInditas.setEnabled(false);
    btnMentes.setEnabled(false);
    lstJatekosok.setModel(jatekosModell);
}
```

```
private void btnInditasActionPerformed(java.awt.event.ActionEvent evt) {
    String nev = txtNev.getText();
    if(nev.isEmpty()){
        JOptionPane.showMessageDialog(this, hibaSzoveg);
    }
    else{
        Jatekos jatekos = new Jatekos(nev);
        vezerlo.jatekInditas(jatekos);
        btnInditas.setEnabled(false);
    }
}

public void jatekVeg(Jatekos jatekos) {
    btnInditas.setEnabled(true);
    eredmenytIr(jatekos);
    btnMentes.setEnabled(true);
}

private void eredmenytIr(Jatekos jatekos) {
    if(jatekosModell.contains(jatekos)){
       jatekosModell.removeElement(jatekos);
    }
    jatekosModell.addElement(jatekos,true);
}
```

Természetesen ahhoz, hogy rendezhető modellbe tudjuk rakni, illetve, hogy működjön a `contains`() metódus, a `Jatekos` osztályt egyrészt `Comparable` típusúvá kell tennünk,

másrészt definiálni kell benne az `equals()+hashCode()` metódust (csak a név azonosságát írjuk elő).

Már csak az adatbázisba írás és az onnan való olvasás, illetve a frame indító metódusának véglegesítése van hátra. Kezdjük az adatbázis-kezeléssel.

A már korábban megbeszélt Dao mintával és a hozzá tartozó interfésszel kezdjük. Ennek most csak a feladatban használt metódusait implementáljuk. Beolvasáskor a `listAll()` metódusra hivatkozunk, kiíratáskor a *create* és *update* műveletet kell megvalósítanunk:

```
public interface Dao <Type, PrimaryKey extends Serializable> {

    public void create(Type t)throws Exception;

    public Type read(PrimaryKey id)throws Exception;

    public void update(Type t)throws Exception;

    public void delete(Type t)throws Exception;

    public List<Type> listAll() throws Exception;
}
```

A beolvasást külön osztályban végezzük, a mentés azonban a vezérlés dolga, de mindkettőnek szüksége van a `JatekosDao` osztály példányára. Azonban elég, ha csak egy példány van belőle, ezért a singleton tervezési minta alapján írjuk meg az osztályt. Mivel oldottunk már meg hasonló feladatot, most magyarázat nélkül közöljük az osztály kódját.

```
public class JatekosDao implements Dao<Jatekos, String> {

    private Connection kapcsolat;
    private static JatekosDao peldany;

    private JatekosDao() {
    }

    public static JatekosDao getPeldany() {
        if(peldany == null){
            peldany = new JatekosDao();
        }
        return peldany;
    }

    public void setKapcsolat(Connection kapcsolat) {
        this.kapcsolat = kapcsolat;
    }
```

```java
@Override
public void create(Jatekos jatekos) throws Exception {
    if (kapcsolat != null) {
        try (Statement utasitasObjektum = kapcsolat.createStatement()) {

            StringBuilder sqlBuilder =
                    new StringBuilder("INSERT INTO JATEKOSOK VALUES ('");
            sqlBuilder.append(jatekos.getNev()).append("',");
            sqlBuilder.append(String.valueOf(jatekos.getPontSzam())).append(")");

            utasitasObjektum.execute(sqlBuilder.toString());
        }
    }
}

@Override
public void update(Jatekos jatekos) throws Exception {
    if (kapcsolat != null) {
        String sqlUtasitas =
                "UPDATE JATEKOSOK set pontszam = ? WHERE nev = ?";

        try (PreparedStatement parameteresUtasitasObjektum
                = kapcsolat.prepareStatement(sqlUtasitas)) {
            parameteresUtasitasObjektum.setInt(1, jatekos.getPontSzam());
            parameteresUtasitasObjektum.setString(2, jatekos.getNev());

            parameteresUtasitasObjektum.executeUpdate();
        }
    }
}

@Override
public List<Jatekos> listAll() throws Exception {
    List<Jatekos> jatekosok = new ArrayList<>();
    String sqlUtasitas = "SELECT * from JATEKOSOK";
    if (kapcsolat != null) {
        try (Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet eredmenyHalmaz =
                            utasitasObjektum.executeQuery(sqlUtasitas)) {

            String nev;
            int pontszam;
            while (eredmenyHalmaz.next()) {
                nev = eredmenyHalmaz.getString("nev");
                pontszam = eredmenyHalmaz.getInt("pontszam");
                jatekosok.add(new Jatekos(nev, pontszam));
            }
        }
    }
    return jatekosok;
}
```

(Az üresen vagy figyelmeztető kivételdobással implementált `read()` és `delete()` metódust nem közöltük.)

Néhány megjegyzés:

- Éles alkalmazásokban a név sohasem lehet elsődleges kulcs, hiszen miért ne lehetne két azonos nevű ember. Játékokban azonban előfordulhat (mint ahogy esetünkben is), hogy csak különböző nevű játékosokat engedünk játszani.

- Elég kényelmetlen egy-egy összetettebb SQL utasítás megírása, hiszen állandóan figyelni kell a string műveletekre is. Ha a `StringBuilder` osztály `append`() metódusát használjuk, akkor ez egy kicsit egyszerűbbé válik.

  Tovább egyszerűsödik a helyzet, ha `Statement` helyett `PreparedStatement` típusú utasításobjektumot használunk, ekkor ugyanis paraméterezett SQL utasítást lehet kiadni. Természetesen mindkét megvalósított műveletet meg lehet oldani mindkét módon, most azért eltérőek, hogy mindkettőre lásson példát, de logikusabb lenne, ha azonos elven írtuk volna meg őket.

- Elvileg még jobb lenne, ha a JDBC helyett a JPA (Java Persistence API) interfészgyűjteményt választanánk, de gyakorlatilag ilyen kis feladatra nem érdemes a JPA-t használni, másrészt sok cég ragaszkodik még a JDBC használatához, ezért fontos annak megismerése is. Ráadásul a feladatgyűjtemény tervezett köre sem mutat túl a JDBC használatán.

Mint korábban szó volt róla, az adatbeolvasást (vagy legelső futtatáskor annak létrehozását) külön szálban oldjuk meg, hogy ne akadályozza meg a felület megjelenítését. Mivel elképzelhető, hogy lassan töltődnek be az adatok, ezért az indító gomb ezek betöltésének végéig inaktív marad.

Az adatbevitelt kezelő osztály (ezt is érdemes singletonként kezelni):

```
public class AdatBetoltoOsztaly extends Thread{

    private final int KEP_SZAM = 10;
    private Vezerlo vezerlo;
    private static AdatBetoltoOsztaly peldany;

    private List<Image> kepek = new ArrayList<>();
    private List<Jatekos> beolvasottJatekosok = new ArrayList<>();

    private AdatBetoltoOsztaly() {
    }

    public static AdatBetoltoOsztaly getPeldany() {
        if(peldany == null){
            peldany = new AdatBetoltoOsztaly();
        }
        return peldany;
    }

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

```
    @Override
    public void run(){

        kepFeltoltes();
        adatBazisMegnyitas();
        vezerlo.betoltottAdatok(kepek, beolvasottJatekosok);
    }

    private void kepFeltoltes(){

        for (int i = 1; i <= KEP_SZAM; i++) {
            kepek.add(new ImageIcon(this.getClass().
                    getResource("/kepek/pillango"+i+".gif")).getImage());
        }
    }

    private void adatBazisMegnyitas() {
        try {
            JatekosDao dao = JatekosDao.getPeldany();
            dao.setKapcsolat(kapcsolodas());
            beolvasottJatekosok = dao.listAll();
        } catch (Exception ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private Connection kapcsolodas() throws ClassNotFoundException, SQLException {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        String url = "jdbc:derby:LepkeDB;create=true;";

        Connection kapcsolat = DriverManager.getConnection(url);

        String sqlVaneMarTabla =
                "select * from SYS.SYSTABLES where tablename = 'JATEKOSOK'";

        try (Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet rs = utasitasObjektum.executeQuery(sqlVaneMarTabla)) {

            if (!rs.next()) {
                String sqlTablaKeszites =
                  " CREATE TABLE JATEKOSOK ( nev varchar(50), pontszam int)";
                utasitasObjektum.execute(sqlTablaKeszites);
            }
        }
        return kapcsolat;
    }
}
```

Ha a szál végzett, jelzi a vezérlő osztálynak. A vezérlő osztály hivatkozott metódusa:

```
public void betoltottAdatok(List<Image> kepek,
                            List<Jatekos> beolvasottJatekosok) {
    this.kepek = kepek;
    this.beolvasottJatekosok = beolvasottJatekosok;
    jatekosPanel.induloListabaIr(beolvasottJatekosok);
}
```

A `JatekosPanel` hivatkozott metódusa:

```
public void induloListabaIr(List<Jatekos> beolvasottJatekosok) {
    for (Jatekos jatekos : beolvasottJatekosok) {
        jatekosModell.addElement(jatekos, true);
    }
    btnInditas.setEnabled(true);
}
```

Természetesen valakinek el kell indítania az adatbeolvasó szálat. Ez a frame dolga lesz néhány egyéb tennivalóval együtt. A frame `main()` metódusából hívjuk meg az `inditas()` metódusát.

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FoFrame().inditas();
        }
    });
}

private void inditas() {
    this.setVisible(true);
    Vezerlo vezerlo = new Vezerlo(jatekPanel1, kijelzoPanel1, jatekosPanel1);
    jatekPanel1.setVezerlo(vezerlo);
    kijelzoPanel1.setVezerlo(vezerlo);
    jatekosPanel1.setVezerlo(vezerlo);
    vezerlo.beallitas();

    ResourceBundle bundle = ResourceBundle.getBundle(BUNDLE);
    vezerlo.bundleBeallitas(bundle);
    String cim = bundle.getString("FRAME_CIM");
    this.setTitle(cim);

    AdatBetoltoOsztaly.getPeldany().setVezerlo(vezerlo);
    AdatBetoltoOsztaly.getPeldany().start();
}
```

A `JatekosPanel` Mentés gombjának megnyomásakor tudjuk elmenteni az adatokat.

```
    private void btnMentesActionPerformed(java.awt.event.ActionEvent evt) {
        List<Jatekos> jatekosok = new ArrayList<>();
//        for (int i = 0; i < jatekosModell.getSize(); i++) {
//            jatekosok.add(jatekosModell.getElementAt(i));
//        }
        jatekosok = jatekosModell.getModell();
        vezerlo.adatMentes(jatekosok);
    }
```

Mivel a `RendezhetoListModel` osztály „saját gyártás”, ezért nem biztos, hogy van benne `getModell()` metódus (egyébként célszerű megírni), ha nincs, akkor a kommentezett sorok is létre tudják hozni a mentendő játékosok listáját. Aki közülük szerepel az adatbázisban, annak módosítani kell a pontszámát, aki nem, azt beszúrni az adatbázisba:

```
    public void adatMentes(List<Jatekos> jatekosok) {
        JatekosDao dao = JatekosDao.getPeldany();
        for (Jatekos mentendoJatekos : jatekosok) {
            try {
                if (beolvasottJatekosok.contains(mentendoJatekos)) {
                    dao.update(mentendoJatekos);
                } else {
                    dao.create(mentendoJatekos);
                    beolvasottJatekosok.add(jatekos);
                }
            } catch (Exception ex) {
                Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Essen szó végre a nyelv beállításáról. Angolul futtatva valahogy így nézzen ki a játék:

Ennek egy része már elő van készítve a korábban bemutatott kódrészletekben. A frame indításakor adjuk át a vezérlőnek a szóban forgó `ResourceBundle` példányt. A vezérlő hivatkozott metódusa:

```
public void bundleBeallitas(ResourceBundle bundle) {
    jatekosPanel.szovegBeallitas(bundle);
    kijelzoPanel.setBundle(bundle);
    Jatekos.setBundle(bundle);
}
```

A `JatekosPanel` hivatkozott metódusa:

```
private String hibaSzoveg;

public void szovegBeallitas(ResourceBundle bundle) {
    lblNevFelirat.setText(bundle.getString("JATEKOSNEV_FELIRAT"));
    hibaSzoveg = bundle.getString("HIBA_SZOVEG");
}
```

A `Jatekos` osztálynak pedig azért kell átadni a `bundle` példányt, hogy a `toString()` metódusában a megfelelő alakot tudja használni:

```
private static ResourceBundle bundle;

@Override
public String toString() {
    String szoveg = bundle.getString("JATEKOS_SZOVEG");
    return nev + " " + pontSzam + " " + szoveg;
}
```

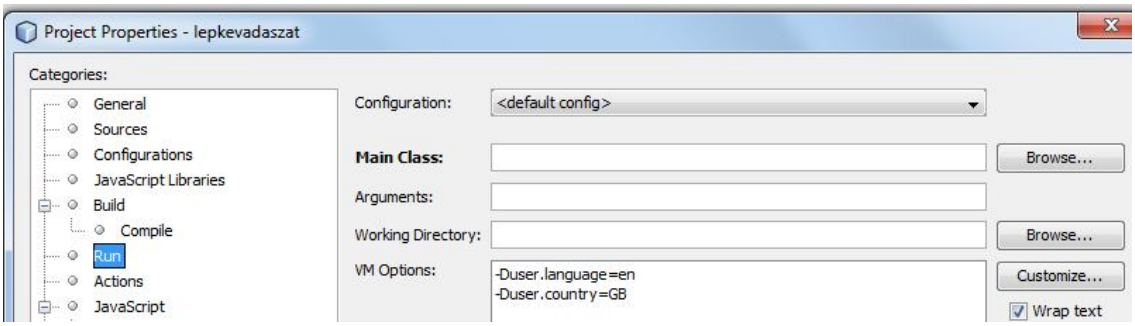
A bundle fájlok tartalma:

Key	default language	hu_HU - magyar (Magyarország)	en_GB - English (United Kingdom)
FRAME_CIM	lepkefogó	Pillangók	Butterflies
btnInditas.text	Indítás	Indítás	Start
JatekosPanel.btnMentes.text	Mentés	Mentés	Save
JatekosPanel.lblEredmenyFelirat.text	Eredmények:	Eredmények:	Results:
JATEKOSNEV_FELIRAT	Játékos neve:	Játékos neve:	Name:
HIBA_SZOVEG	Nem adott meg nevet	Kérem a nevét	Please enter your name
KijelzoPanel.lblPontszam.text	Pontszám:	Pontszám:	Score:
KijelzoPanel.lblIdo.text	Hátralévő idő: %d sec	Hátralévő idő: %d sec	Remaining time: %d sec
JATEKOS_SZOVEG	pont	pont	points

(Így nézhető meg: bundle.properties fájl: jobb egérgomb, open – de egyenként is lehet szerkeszteni őket.)

Az igazi megoldás az lenne, amit már a korábbi lokalizációs példában is csináltunk, vagyis az, hogy egy menü vagy rádiógombok segítségével a program futásakor állítjuk be a nyelvet, de fixen is beállítható a virtuális gépnek adott paranccsal.

NetBeans-ben: projektnév, jobb egérgomb, Properties és:



A használt parancs: -Duser.language=en -Duser.country=GB

Persze, ha más nyelven szeretné, akkor értelemszerűen más ☺.

Végül, de nem utolsósorban egy egyszerű kis teszt. (Ne felejtse, a tesztelés mindig nagyon fontos része a programírásnak.)

```
public class Teszt {

    private Image kep;
    private int kepX, kepY;
    private int kepSzelesseg, kepMagassag;
    private long ido;
    private int dx;
    private int dy;

    private Vezerlo vezerlo;
    private Pillango pillango;

    public Teszt() {
    }

    @Before
    public void setUp() {
        kepX = 10;
        kepY = 10;
        kepSzelesseg = 100;
        kepMagassag = 100;
        dx = 2;
        dy = 1;
        pillango = new Pillango(kep, kepX, kepY, kepSzelesseg, kepMagassag,
                        ido, dx, dy, vezerlo);
    }
```

```
    @Test
    public void pillangoTeszt() {
        pillango.megfogtak(dx, dy);
        assertFalse(pillango.isElkaptak());

        pillango.megfogtak(kepX + dx, kepY + dy);
        assertTrue(pillango.isElkaptak());

        Jatekos jatekos = new Jatekos("nev");
        jatekos.pontotKap();
        assertEquals(jatekos.getPontSzam(), 1);

        jatekos.pontotKap();
        assertEquals(jatekos.getPontSzam(), 2);
    }
}
```