# Akvárium

Írjunk egy akvárium-szimuláló programot.

Induló felület:
(500 × 400 pixel)


Rákattintva ez a felület eltűnik, megjelenik az akvárium:
(belső méret: 850 × 460, ebből a vezérlő rész 150 pixel)


A halneveket adatbázisban tároljuk (ez a legelső futtatáskor jön létre).

Minden halpéldánynak lesz neve, megadható a bal- és jobboldali profilképük és a képméret.

A képek a *kepek* mappában találhatók a nevek sorrendjében, méretük véletlenül generálható az adott határok között.

(Az igazi az lenne, ha a képneveket a nevekkel együtt az adatbázisban tárolnánk, ezt majd próbálja meg önállóan módosítani.)

A „Vízbedobás” gomb hatására a listából kiválasztott halak (egyszerre többet is lehet választani) bekerülnek a vízbe, és ott elkezdenek úszkálni. Egyúttal a listából kikerül a nevük.

Az úszás egyelőre ezt jelenti: véletlen sebességgel véletlenszerűen indulnak balra vagy jobbra, és ha az „akvárium” falához érnek, akkor visszafordulnak.

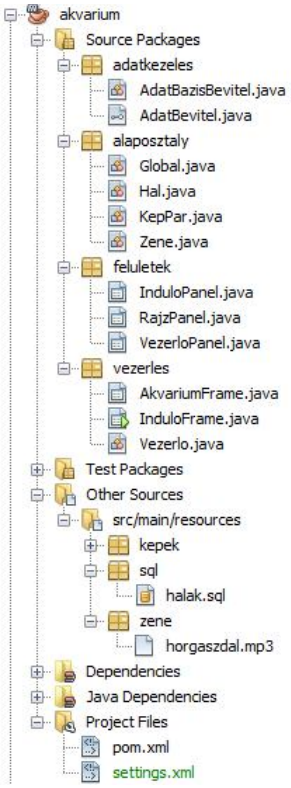
Oldjuk meg azt is, hogy a hangszóró gombra kattintva szólaljon meg egy háttérzene, majd ismét megnyomva hallgasson el.

A vízre kattintva megijednek a halak, és vagy egyszerre mind megáll, vagy ha éppen álltak, akkor egyszerre mind nekiindul. Mivel a halak erősen társas lények, ezért az újonnan vízbe dobott hal alkalmazkodik a többihez, és ha ők állnak, akkor az új is csak akkor kezd el mozogni, amikor a többi, ha mozognak, akkor ő is mozog.

Ha mégsem csak gyönyörködni szeretnénk a halakban, akkor, ha eltalálunk egy halat, azt tüntessük el az akváriumból. Úgy is módosíthatjuk, hogy az eltalált halat majd ismét vissza lehessen dobni, vagyis kerüljön vissza a listába.

Végül: a kicsit „élethűbb” mozgás kedvéért oldjuk meg azt, hogy időnként ne csak akkor forduljon meg egyik-másik, amikor falhoz ér, hanem akkor, amikor kedve tartja, és engedjünk meg a mozgásukban némi le-föl hullámzást is.

**Megoldásrészletek**




A megoldáshoz Maven projektet készítünk. A projekt szerkezete:

A *pom.xml* fájl a projektnév kivételével ugyanaz, mint a korábbi Maven alkalmazás esetén (megtalálható az adatfájlokat tartalmazó mappában).

Eddig (és a későbbiekben is) úgy oldottuk meg a feladatokat, hogy a használt konstans értékeket az aktuális osztály elejére gyűjtöttük. Most arra is látunk példát, hogy a programban használt összes konstans értéket egy közös, `Global` nevű osztályba gyűjtjük. (**Fontos**: erre az osztályra csak vezérlő jellegű osztályokból hivatkozhatunk, alap-osztályokból soha.)

A `Global` osztály tartalma:

```
public class Global {

    public static final String SQL_ELERES = "/sql/halak.sql";
    public static final String INDULO_KEP_ELERES = "/kepek/indulo.jpg";
    public static final String HATTERKEP_ELERES = "/kepek/hatter.jpg";
    public static final String KEPFAJL_ELERES = "/kepek/hal";
    public static String IKON_ELERES = "/kepek";
    public static String ZENEFAJL_ELERES = "/zene/horgaszdal.mp3";

    public static final int ALSO_KEPMERET = 40,
                            FELSO_KEPMERET = 70;

    // a képnek ez a felső aránya lesz a "levegő", itt nem úszhatnak a halak.
    public static final double KEP_MAGASSAG_ARANY = 0.2;
    public static final double FELSO_IDO = 30;
    public static final double ALSO_IDO = 10;
}
```

**1. Felületváltás:**

Most két különböző frame-t hozunk létre, de csak az `InduloFrame` osztályban hagyjuk meg a `main()` metódust, a másikból kitöröljük.

Az `InduloFrame` tetejére kerül az `InduloPanel`, az `AkvariumFrame` tetejére pedig egymás mellé a `VezerloPanel` és a `RajzPanel`.

Az induló panel üres, majd csak egy háttérrajz lesz rajta. A vezérlő panelre kerülnek fel a szükséges gombok és a listafelület, a rajzolásra szolgáló panel szintén üres. A frame-re való felhúzáskor ne felejtse el border layout-ra állítani a frame-t, és óvatosan rakja rá a két panelt (vagyis úgy, hogy ne foglalja el valamelyikük a teljes felületet).

Az induló frame kódjában a méret és cím beállításán, illetve a generált részen kívül semmi nincs. A frame-ket célszerű átméretezhetetlenre állítani, de csak akkor, ha már fut a program!

Az induló panelre rákerül a rajz, és figyelni kell a felületre való kattintás hatását. Az osztály kódja (generált részletek nélkül):

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

    private Image hatterKep =
            new ImageIcon(this.getClass().
                          getResource(Global.INDULO_KEP_ELERES)).getImage();
    public InduloPanel() {
        initComponents();
    }

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

    private void formMouseClicked(java.awt.event.MouseEvent evt) {
        AkvariumFrame akvariumFrame = new AkvariumFrame();
        akvariumFrame.setVisible(true);
        akvariumFrame.indit();
        // lekéri azt a frame-t, amelyre felraktuk ezt a panelt
        JFrame frame = (JFrame) SwingUtilities.getRoot(this);
        frame.setVisible(false);
    }
```

**2. Adatbevitel:**

Beolvasáskor a hal nevén kívül megadjuk még a hozzá tartozó két képet és a képek méretét. Vagyis a `Hal` osztály konstruktorának ezeket az adatokat kell paraméterként tartalmaznia. Mivel fontos, hogy ugyanazt a halat lássuk jobbra vagy balra haladva, ezért lényeges, hogy a

hozzá tartozó két kép mindig együtt szerepeljen. Emiatt ezeket egy külön osztály megfelelő példányaiban tároljuk majd.

A `Hal` osztály konstruktora és a hozzá tartozó mező-deklarációk:

```
    private String nev;
    private int kepSzelesseg, kepMagassag;
    private KepPar kepPar;

    public Hal(String nev, int kepSzelesseg, int kepMagassag, KepPar kepPar) {
        this.nev = nev;
        this.kepSzelesseg = kepSzelesseg;
        this.kepMagassag = kepMagassag;
        this.kepPar = kepPar;
    }
```

A konstruktorban hivatkozott `KepPar` osztály tartalmazza a halhoz tartozó kép-párokat:

```
public class KepPar {

    private Image balKep;
    private Image jobbKep;

    public KepPar(Image balKep, Image jobbKep) {
        this.balKep = balKep;
        this.jobbKep = jobbKep;
    }

    public Image getBalKep() {
        return balKep;
    }

    public Image getJobbKep() {
        return jobbKep;
    }
}
```

Az előkészületek megtétele után jöhet a beolvasás. Mivel ez a feladat csak beolvasásra használja az adatbázist, ezért most nem alkalmazzuk a DAO mintát, hanem csak egy egyszerű interfészből indulunk ki, de egy esetleges továbbfejlesztés során könnyen kicserélhető vagy bővíthető ez az interfész úgy, hogy kövesse ezt a mintát.

Az adatbeolvasó interfész:

```
public interface AdatInput {
    public List<Hal> halLista() throws Exception;
}
```

Olvasáskor az adatbázisból beolvassuk a neveket, majd minden egyes névhez hozzárendelünk egy-egy képpárt. Logikusabb lenne az adatfájlban a névhez tartozó képneveket is megadni (ekkor ugyanis garantáltan ugyanaz a kép tartozik egy adott névhez), de most (főleg azért, hogy ilyenre is lásson példát), a képeket sorszám szerint különböztetjük meg, és a beolvasás sorrendjében rendeljük a halakhoz. Ehhez nyilván kell majd egy változó, amelyben a sorszámot tároljuk, és evvel együtt hivatkozunk a fájlnévre.

Kicsit vitatható az a megoldás, amelyet most választunk, vagyis az, hogy a képfájlok elérési útvonalát tartalmazó stringbe a fájlnév közös részét is belevesszük, de kényelmessé teszi a munkát. A megoldás hátránya: a változónév nem, vagy csak „kacifántosan” elnevezve takarja pontosan a tartalmat.
Előnye: bármely olyan esetben használható, amikor kezdőszöveg+sorszám+BAL/JOBB szerkezetű a fájlnév.

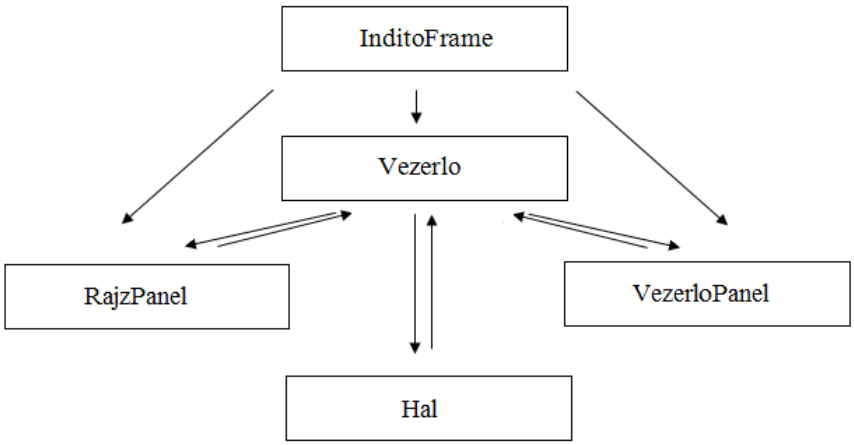
A képméreteket adott határok között véletlenszerűen adjuk meg.

Az adatbeviteli osztály kódja:

```
public class AdatBazisBevitel implements AdatBevitel {

    private Connection kapcsolat;

    private int alsoMeret, felsoMeret;
    private String kepFajlEleresKezdoSzoval;

    public AdatBazisBevitel(Connection kapcsolat, int alsoMeret, int felsoMeret,
                            String kepFajlEleresKezdoSzoval) {
        this.kapcsolat = kapcsolat;
        this.alsoMeret = alsoMeret;
        this.felsoMeret = felsoMeret;
        this.kepFajlEleresKezdoSzoval = kepFajlEleresKezdoSzoval;
    }

    @Override
    public List<Hal> halListaBevitel() throws Exception {
        List<Hal> halak = new ArrayList<>();

        String sqlCommand = "select * from HALAK";

        int sorSzam = 0;
        try (Statement utasitasObjektum = kapcsolat.createStatement();
                ResultSet eredmenyHalmaz
                                = utasitasObjektum.executeQuery(sqlCommand)) {

            String nev;
            int kepSzel, kepMag;
            Image balKep, jobbKep;
```

```
            while (eredmenyHalmaz.next()) {
                nev = eredmenyHalmaz.getString("nev");
                balKep = new ImageIcon(this.getClass().
                        getResource(kepFajlEleresKezdoSzoval
                                    + sorSzam + "_BAL.png")).getImage();
                jobbKep = new ImageIcon(this.getClass().
                        getResource(kepFajlEleresKezdoSzoval
                                    + sorSzam + "_JOBB.png")).getImage();
                kepSzel = (int) (Math.random()*(felsoMeret - alsoMeret) +
                                                                alsoMeret);
                kepMag = (int) (Math.random()*(felsoMeret - alsoMeret) +
                                                                alsoMeret);
                halak.add(new Hal(nev, kepSzel, kepMag,
                                  new KepPar(balKep, jobbKep)));

                sorSzam++;
            }
        }
        return halak;
    }
}
```

A feladatmegoldás során (a korábban vett megoldásokhoz hasonlóan) az újrahasznosíthatóság is célunk, ezért fontos, hogy az egyes osztályok minél függetlenebbek legyenek egymástól, vagyis mindegyik csak a vezérlővel tartson kapcsolatot.




A beolvasást a vezérlő osztály indítja el, majd megkéri a vezérlőpanelt, hogy írja ki a lista-felületre a beolvasott adatokat. Ezért a vezérlőnek ismernie kell a vezérlőpanelt. De a panelnek is ismernie kell a vezérlőt, hiszen jeleznie kell neki, hogy mely halakat választottuk ki.

A rajzpanel azt rajzolja majd, amit a vezérlő mond neki, ezért a rajzpanelnek is ismernie kell a vezérlőt. Ez az ismeretség is kölcsönös, mert a vezérlőnek is ismernie kell a rajzpanelt, hiszen időnként frissítésre kéri.

A vezérlő kezeli a kiválasztott halakat, ezért nyilván ismernie kell őket, ugyanakkor a hal példányoknak is ismerniük kell a vezérlőt, hiszen időnként őt kérik majd frissítésre.

**Megjegyzés:** A most tárgyalt megoldás nem ennyire tiszta szerkezetű, ugyanis a vezérlőpanel listamodellje `Hal` típusú példányokat tartalmaz, vagyis a megoldásunkban van egy „titkos" kapcsolat a vezérlőpanel és a `Hal` osztály között. Ennek kiküszöböléséről egy külön pdf fájlban lesz szó (*tovabbi_halas_variaciok.pdf*).

Az `AkvariumFrame indit()` metódusában (ezt hívtuk meg az induló panelre kattintva) ismertetjük össze egymással az egyes példányokat, és itt hívjuk meg a vezérlő `indit()` metódusát is. (Mivel a szál példányokat a példány `start()` metódusával indítjuk, ezért most szerencsésebbnek tűnik, hogy az általunk írt indító metódusokat ne start néven emlegessük.)

```
public void indit() {
    Vezerlo vezerlo = new Vezerlo(rajzPanel1, vezerloPanel1);
    vezerloPanel1.setVezerlo(vezerlo);
    rajzPanel1.setVezerlo(vezerlo);

    vezerlo.indit();
}
```

A `Vezerlo` osztály az `indit()` metódusig (indításkor állítjuk be a rajzfelület méretét is):

```
public class Vezerlo {

    private String CHAR_SET = "UTF-8";

    // Szálpéldányok kezelésére ez a listafajta biztonságos
    private List<Hal> halak = new CopyOnWriteArrayList<>();

    private RajzPanel rajzPanel;
    private VezerloPanel vezerloPanel;

    private Zene zene;

    public Vezerlo(RajzPanel rajzPanel, VezerloPanel vezerloPanel) {
        this.rajzPanel = rajzPanel;
        this.vezerloPanel = vezerloPanel;
    }

    public void indit() {
        Hal.setAkvariumSzelesseg(rajzPanel.getWidth());
        Hal.setAkvariumMelyseg(rajzPanel.getHeight()-Global.FELSO_HAL_KEPMERET);
        Hal.setVizMagassag((int) (rajzPanel.getHeight()
                                            *Global.KEP_MAGASSAG_ARANY));
        zene = new Zene();
        adatBeolvasas();
    }
```

Adatbeolvasás:

```
public void adatBeolvasas() {
    try {
        List<Hal> beolvasottHalak;
        AdatBevitel adatBevitel =
                new AdatBazisBevitel(kapcsolodas(), Global.ALSO_HAL_KEPMERET,
                                                    Global.FELSO_HAL_KEPMERET,
                                                    Global.KEPFAJL_ELERES);
        beolvasottHalak = adatBevitel.halListaBevitel();
        vezerloPanel.ittVannakaBeolvasottAdatok(beolvasottHalak);
    } 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:AkvariumDB;create=true;";

    Connection kapcsolat = DriverManager.getConnection(url);

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

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

        if (!rs.next()) {
            adatBazisLetrehozas(utasitasObj);
        }
    }
    return kapcsolat;
}

private void adatBazisLetrehozas(Statement utasitasObj) throws SQLException {
    try(InputStream ins = this.getClass().getResourceAsStream(Global.SQL_ELERES);
            Scanner sc = new Scanner(ins, CHAR_SET)){
        String sor;
        while(sc.hasNextLine()){
            sor = sc.nextLine();
            System.out.println(sor);
            utasitasObj.execute(sor);
        }
    } catch (IOException ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Vagyis ha még nincs ilyen adatbázis, akkor létrehozzuk a megadott sql fájl alapján, ha van, akkor csak beolvassuk belőle az adatokat.

A vezérlőpanel hivatkozott metódusa (és néhány egyéb, a felületkialakításhoz szükséges rész):

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

    private DefaultListModel<Hal> halModell;
    private final Icon hangBe =
        new ImageIcon(getClass().getResource(Global.IKON_ELERES+"/hangbe.png"));
    private Icon hangKi =
        new ImageIcon(getClass().getResource(Global.IKON_ELERES+"/hangki.png"));
    private int gombMeret = 30;

    public VezerloPanel() {
        initComponents();
        btnHang.setSize(gombMeret, gombMeret);
        btnHang.setLocation(this.getWidth()/2 - gombMeret/2,
                            this.getHeight()-2*gombMeret);
        btnHang.setIcon(hangBe);
    }

    private Vezerlo vezerlo;

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

    public void ittVannakaBeolvasottAdatok(List<Hal> beolvasottHalak) {
        halModell = new DefaultListModel<>();
        for (Hal hal : beolvasottHalak) {
            halModell.addElement(hal);
        }
        lstHalak.setModel(halModell);
    }
```

**Megjegyzés**: Remélhetőleg nem csak olvasta, hanem csinálta is a megoldást. Ha eddig eljutott, akkor tapasztalhatja, hogy – különösen a legelső futtatáskor, de később is – nagyon lassan töltődnek be az adatok, addig szinte le van fagyva az alkalmazás (még az akvárium felülete sem jelenik meg). Ezt a problémát orvosoljuk majd, de csak a megoldás legvégén.

### 3. Zene

A zenekezelés továbbra sem elsőrendű célunk, inkább csak színesítésként szerepel. A kódrészletek ismertetésén kívül nem is foglalkozunk vele.

Zenelejátszáshoz a javazoom külső csomagot használjuk, amit a *pom.xml* fájl alapján rendelünk hozzá a programunkhoz.

A szükséges kódrészletek (getterek nélkül):

```
public class Zene extends Thread{

    private String zenefajlEleres;
    private InputStream stream;
    private Player player;
    private boolean szolhat = true;

    public void setZeneFajlEleres(String zenefajlEleres) {
        try {
            this.zenefajlEleres = zenefajlEleres;
            stream = this.getClass().getResourceAsStream(zenefajlEleres);
            player = new Player(stream);
        } catch (JavaLayerException ex) {
            Logger.getLogger(Zene.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void run(){
        try {
            player.play();
        } catch (JavaLayerException ex) {
            Logger.getLogger(Zene.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void leall() {
        player.close();
    }
```

A vezérlőpanel hang gombjához tartozó esemény:

```
    private void btnHangActionPerformed(java.awt.event.ActionEvent evt) {
        if(btnHang.getIcon().equals(hangBe)){
            vezerlo.zeneBekapcsolas();
            btnHang.setIcon(hangKi);
        }
        else{
            vezerlo.zeneKikapcsolas();
            btnHang.setEnabled(false);
        }
    }
```

A vezérlő osztály hivatkozott metódusai:

```
     public void zeneBekapcsolas() {
         if(!zene.isAlive()){
             zene.setZeneFajlEleres(Global.ZENEFAJL_ELERES);
             zene.start();
         }
     }

     public void zeneKikapcsolas() {
         zene.leall();
     }
```

Bár korábban is volt már szó zenelejátszásról, talán most világosabb, hogy mi történik a háttérben: a zenét külön szálban indítjuk, különben addig semmi egyebet sem csinálhatnánk, amíg be nem fejeződik. Egy szálat csak egyszer lehet elindítani, ezért vizsgáljuk bekapcsoláskor, hogy él-e már a szál. Természetesen azt is meg lehetne oldani, hogy a gomb hatására ki-be lehessen kapcsolgatni, de mint szó volt róla, most nem a zenekezelésen van a hangsúly, ezért kikapcsolás után inaktívvá tesszük a gombot. (Ha a múltkori feladatban nem tette, akkor most próbálja ki, mi történik, ha a `start()` helyett a zene példány `run()` metódusát hívja meg – ekkor nem külön szálban fut a metódus.)

**4. Halacskázás:**

Vízbedobás:

A halak vízbedobása elég egyszerű: a vezérlőpanelen a kiválasztott halakat egyenként ki kell törölni a modellből, és ugyanakkor megkérni a vezérlőt, hogy rakja vízbe őket.

```
private void btnVizbeActionPerformed(java.awt.event.ActionEvent evt) {
    List<Hal> halak = lstHalak.getSelectedValuesList();
    for (Hal hal : halak) {
        vezerlo.vizbeDob(hal);
        halModell.removeElement(hal);
    }
}
```

A vízbedobás a következőt jelenti: a vezérlő véletlenszerűen generálja a hal helyét (az akváriumon, sőt, a vízmagasságon belül), és ugyancsak véletlenszerűen generálja a mozgásához szükséges időt. (Minden mozdulat után ennyit pihen majd a hal.) Azt most beégetjük, hogy fele-fele eséllyel indul jobbra vagy balra. Beállítjuk a hal szükséges adatait, majd elindítjuk. Mivel mozognak, ezért szálként kezeljük a hal példányokat (kódját ld. kicsit később), és ezt a szálat indítjuk el.

A feladat szerint a később bedobott halak alkalmazkodnak majd a többiekhez, vagyis ha azok úsznak, akkor ő is, ha épp nem úsznak, akkor ő sem. A legelső hal viszont azonnal úszni kezd, csak későbbi események hatására változik a halak viselkedése.

Ezeket a halakat ki is kell rajzolni, illetve frissíteni a képernyőképet.

A `Vezerlo` osztály ide vonatkozó metódusai:

```
public void rajzol(Graphics g){
    for (Hal hal : halak) {
        hal.rajzol(g);
    }
}

public void frissit() {
    rajzPanel.repaint();
}

public void vizbeDob(Hal hal) {
    int kepX = (int) (Math.random()
            * (Hal.getAkvariumSzelesseg() - hal.getKepSzelesseg()));
    int kepY = (int) (Math.random()
            * (Hal.getAkvariumMelyseg() - Hal.getVizMagassag()
                                        - hal.getKepMagassag())
            + Hal.getVizMagassag());
    long ido = (long) (Math.random() * (Global.FELSO_IDO - Global.ALSO_IDO)
            + Global.ALSO_IDO);
    int lepesKoz = (Math.random() < 0.5) ? 1 : -1;

    hal.beallit(kepX, kepY, ido, true, lepesKoz, this);

    if (!halak.isEmpty()) {
        hal.setUszik(halak.get(0).isUszik());
    } else {
        hal.setUszik(true);
    }
    halak.add(hal);
    frissit();
    hal.start();
}
```

A rajzpanelnek nagyon egyszerű dolga van: ki kell rajzolnia azt, amit a vezérlő mond, plusz persze a háttérképet:

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

    private Vezerlo vezerlo;

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

    private Image hatterKep =
            new ImageIcon(this.getClass().
                                getResource(Global.HATTERKEP_ELERES)).getImage();

    public RajzPanel() {
        initComponents();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(hatterKep, 0, 0, this.getWidth(), this.getHeight(), null);
        if(vezerlo != null) vezerlo.rajzol(g);
    }
```

Beszéljük meg végre a `Hal` osztály részleteit is!

A hal példányok létrehozásakor már szó volt az osztály konstruktoráról, de most meg kellene tárgyalni a többi részét is.

A hal feladatai:

- Megmondja, hogyan lehet kirajzolni.
- Azt is megmondja, hogyan lehet kiíratni (`toString()`)
- Mivel mozog, ezért szálként kezelendő, és meg kell írni a `run()` metódusát.
- Vizsgálni kell, hogy eltalálták-e.
- Lehetőséget kell adnia a „működésváltásra”, vagyis arra, hogy hol ússzon, hol ne.

Mivel az összes hal ugyanabban az akváriumban úszkál, ezért az erre vonatkozó méretek statikusak.

Ezek után a `Hal` osztály (setterek/getterek nélkül):

```java
public class Hal extends Thread{

    private String nev;
    private int kepX;
    private int kepY;
    private int kepSzelesseg, kepMagassag;

    private KepPar kepPar;
    private Image kep;

    private long ido;
    private boolean aktiv;
    private int lepesKoz;
    private Vezerlo vezerlo;
    private boolean uszik;

    private static int akvariumSzelesseg;
    private static int akvariumMelyseg;
    // ilyen magasságig úszhatnak a halak
    private static int vizMagassag;

    public Hal(String nev, int kepSzelesseg, int kepMagassag, KepPar kepPar) {
        this.nev = nev;
        this.kepSzelesseg = kepSzelesseg;
        this.kepMagassag = kepMagassag;
        this.kepPar = kepPar;
    }

    public void beallit (int kepX, int kepY, long ido, boolean aktiv,
                         int lepesKoz, Vezerlo vezerlo) {
        this.kepX = kepX;
        this.kepY = kepY;
        this.ido = ido;
        this.aktiv = aktiv;
        this.lepesKoz = lepesKoz;
        this.vezerlo = vezerlo;
        kepBeallitas();
    }

    private void kepBeallitas() {
        kep = (lepesKoz < 0) ? kepPar.getBalKep() : kepPar.getJobbKep();
    }

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

```java
@Override
public void run() {
    while (aktiv) {
        varakozik();
        mozdul();
        frissit();
        pihen();
    }
}

private void mozdul() {
    kepX += lepesKoz;
    if(kepX >= akvariumSzelesseg - this.kepSzelesseg || kepX <= 0){
        lepesKoz = -lepesKoz;
        kepBeallitas();
    }
}

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

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

```java
public boolean eltalaltak(int x, int y) {
    return kepX <= x && x <= kepX + kepSzelesseg &&
           kepY <= y && y <= kepY + kepMagassag;
}

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

```java
private synchronized void varakozik(){
    if(!uszik){
        try {
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Hal.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

A rajzpanelre kattintva tudunk eltalálni egy halat, vagyis a szükséges metódus:

```
    private void formMousePressed(java.awt.event.MouseEvent evt) {
        vezerlo.talalatVizsgalat(evt.getX(), evt.getY());
    }
```

A `Vezerlo` osztály ide vonatkozó metódusa:

```
    public void talalatVizsgalat(int x, int y) {
        boolean eltalaltakEgyet = false;
        for (Hal hal : halak) {
            if(hal.eltalaltak(x,y)){
                kiszed(hal);
                eltalaltakEgyet = true;
                break;
            }
        }
        if(!eltalaltakEgyet){
            for (Hal hal : halak) {
                hal.mukodesValtas();
            }
        }
    }

    private void kiszed(Hal hal) {
        hal.setAktiv(false);
        halak.remove(hal);
        frissit();
    }
}
```

Ha a halak cikázó mozgását is szimulálni szeretnénk, akkor kicsit módosítani kell a `Hal` osztályt, például így (másfajta mozgást is kitalálhat).

Az osztályba bekerülnek ezek a mezők:

```
    private int lepesKozX, lepesKozY;
    private int eredetiY;

    private int hullamHatar;
    private static double fordulasHatar;
```

A `fordulasHatar` változó a vezérlő `indit()` metódusában kaphat értéket, pl. 0,01-et – ekkora eséllyel fordul meg hamarabb. (A megadott érték 1% esélyt jelent.)

A `beallit()` metódusba bekerül még ez a két sor:

```
        this.eredetiY = kepY;
        this.hullamHatar = 2*kepMagassag;
```

(Lehet más is a hullámhatár.)

A módosított `run()` és `mozdul()` metódus:

```
@Override
public void run() {
    while (aktiv) {
        varakozik();
        mozdul();
        frissit();
        pihen();
        if(Math.random() < fordulasHatar) {
            lepesKozX = -lepesKozX;
            kepBeallitas();
        }
    }
}

private void mozdul() {

    kepX += lepesKozX;
    kepY += lepesKozY;
    if(kepX >= akvariumSzelesseg - this.kepSzelesseg || kepX <= 0){
        lepesKozX = -lepesKozX;
        kepBeallitas();
    }

    if (kepY > (eredetiY + hullamHatar - kepMagassag) ||
                kepY > akvariumMelyseg - kepMagassag ||
                kepY < eredetiY - hullamHatar ||
                kepY < vizMagassag) {
        lepesKozY = -lepesKozY;
    }
}
```

**Korábbi ígéret – javított beolvasás:**

Ahogy korábban megbeszéltük, illetve a saját megoldásában is tapasztalhatta, az adatbetöltés idejére leáll az élet, pedig most kevés adattal dolgozunk. Képzelje el, mi lenne, ha jókora adatbázissal dolgoznánk.

Valószínűleg sejti már a megoldást, hiszen a jelenség nagyon hasonlít a zenelejátszáskor megbeszéltekhez. Ott az volt a megoldás, hogy külön szálban indítottuk a zenelejátszást. Ez az ötlet most is jónak tűnik. Az adatbeolvasást egy külön szálban már elkezdhetjük az első felület megjelenésekor, a másodikra váltva már csak a beolvasott adatok megjelenítésével kell foglalkozni. Persze, sok adat esetén nem kizárt, hogy az első felületre kattintva (ekkor kellene a másodikra áttérni) még nincsenek meg az adatok. Az ilyen hibás váltást egyszerűen, egy várakozásra kérő üzenettel ki lehet küszöbölni.

Még egy dolgot kell végiggondolni: A beolvasást a vezérlő osztály irányítja, és ha készen van, akkor jeleznie kell az induló panelnek, hogy most már hatásosan kattinthatunk a felületére. Ehhez viszont a vezérlőnek ismernie kell az induló panel példányt, ami viszont az induló frame-n jött létre. Az nyilván nem lehet jó megoldás, hogy mindkét frame-n létrehozunk egy-egy vezérlő példányt, hiszen ugyanazt a példányt kellene használnia mindkét frame-nek. A singleton tervezési minta segítségével azonban könnyedén megoldható, hogy a vezérlőből csak egyetlen példányt lehessen létrehozni

A `Vezerlo` osztályt tehát átalakítjuk úgy, hogy singleton legyen, illetve implementálja a `Runnable` interfészt. Az osztályban végrehajtott módosítások:

```
public class Vezerlo implements Runnable{

    private static Vezerlo peldany;
    private InduloPanel induloPanel;

    private Vezerlo() {
    }

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

    @Override
    public void run() {
        adatBeolvasas();
        induloPanel.aktivalas();
    }

    public void setInduloPanel(InduloPanel induloPanel) {
        this.induloPanel = induloPanel;
    }
```

Az osztály további módosításai:

Az `adatBeolvasas()` metódusban korábban lokálisan deklarált `beolvasottHalak` listát most mezőként deklaráljuk; a metódusból kivesszük a panelre írást, vagyis kitöröljük a `vezerloPanel.ittVannakaBeolvasottAdatok(beolvasottHalak);` hivatkozást:

```
    private List<Hal> beolvasottHalak;

    public void adatBeolvasas() {
        ...
            beolvasottHalak = adatBevitel.halListaBevitel();
//            vezerloPanel.ittVannakaBeolvasottAdatok(beolvasottHalak);
```

Az `indit()` metódusban pedig az adatbeolvasás helyett a panelre írást hívjuk meg:

```
    public void indit() {
        ...
//          adatBeolvasas();
          vezerloPanel.ittVannakaBeolvasottAdatok(beolvasottHalak);
```

Az `InditoFrame` osztálynak most egy kis teendője lesz, a `main()` metódusából meghívott indító metódus:

```
    private void indit() {
        this.setVisible(true);
        Vezerlo vezerlo = Vezerlo.getPeldany();
        vezerlo.setInduloPanel(induloPanel1);
        Thread adatbeolvasoSzal = new Thread(vezerlo);
        adatbeolvasoSzal.start();
    }
```

(Természetesen az `AkvariumFrame` osztályban is `Vezerlo.getPeldany()` módon hivatkozunk a vezérlő példányra.)

Végül az `InduloPanel` teendői is megváltoztak. Ha rákattintunk, de még nincs kész a beolvasás, akkor egy figyelmeztető üzenetet kell adnia, egyébként pedig lehetővé tenni a felületváltást. Arról, hogy készen van-e az olvasás, a vezérlő osztály `run()` metódusa értesíti a panelt.

Az `InduloPanel` osztályban végrehajtott módosítások:

```
    private boolean aktiv;

    public void aktivalas() {
        aktiv = true;
    }

    private void formMouseClicked(java.awt.event.MouseEvent evt) {
        if (!aktiv) {
            JOptionPane.showMessageDialog(this,
                                    "Várjon, még nem töltődtek be az adatok");
        } else {
            AkvariumFrame akvariumFrame = new AkvariumFrame();
            akvariumFrame.setVisible(true);
            akvariumFrame.indit();
            // lekéri azt a frame-t, amelyre felraktuk ezt a panelt
            JFrame frame = (JFrame) SwingUtilities.getRoot(this);
            frame.setVisible(false);
        }
    }
```

**Megjegyzés**: A most bemutatott alapötlet jó, de komolyabb feladatok megoldásához érdemes kipróbálni a Java `SwingWorker` osztályát.

https://docs.oracle.com/javase/8/docs/api/javax/swing/SwingWorker.html

http://www.javacreed.com/swing-worker-example/