Halas feladat – szálkezelő részletek

Feladat:

Az ábrán látható felület mérete: 850*500, a vezérlő rész 150 pixel széles.
Induláskor minden üres, csak a háttérkép látszódik.

Az „Adatbetöltés” gomb hatására kerülnek be a halak a rendszerbe.

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.

2. A vízre való kattintáskor ez lehet:
a/ Ha eltaláltunk egy halat, akkor kikerül a vízből,
b/ ha nem, akkor viszont az összesnek megváltozik a mozgása (megáll, vagy újraindul).

Megoldásrészletek

Ide most ezek a részletek kerülnek:
A részletek sorrendje:

Vezérlőpanel
    a vízbedobás gomb hatása

Rajzpanel
    rajzolás és felületre kattintás

Vezérlő osztály
    az ide vonatkozó részletek

Vezérlőpanel, a halak vízbe dobása: a kiválasztott halakat egyenként ki kell törölni a modellből, és ugyanakkor megkérni a vezérlőt, hogy rakja vízbe.

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

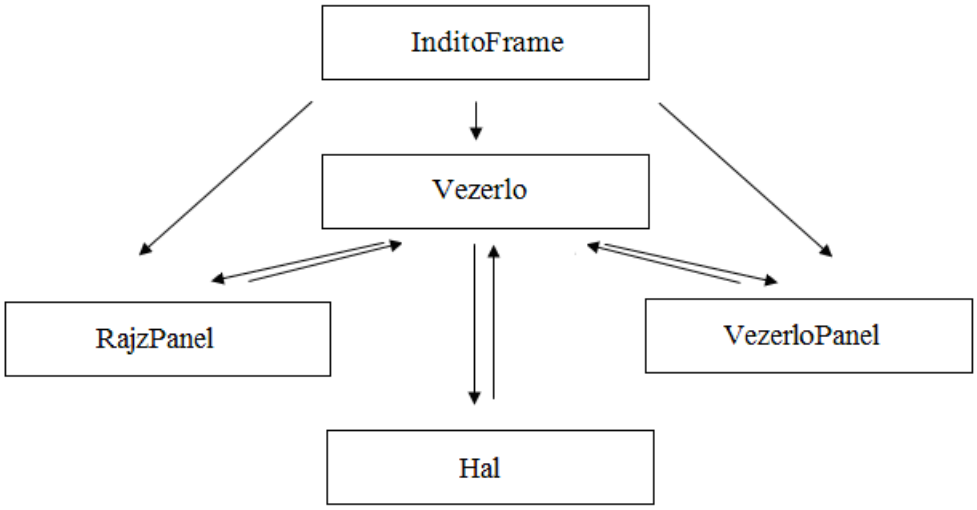
Már csak az a kérdés, honnan kap értéket a `vezerlo` példány.
Ez a `Vezerlo` osztály egy példánya, és setterrel kaphat értéket:

```
private Vezerlo vezerlo;

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

Beszéljük meg azt is, hogy hol hívjuk meg a settert. De hogy ne aprózzuk el teljesen, gondoljuk végig a teljes szerkezetet.

A feladatmegoldás során az újrahasznosíthatóság is célunk, ezért úgy akarjuk megoldani, hogy az egyes osztályok minél függetlenebbek legyenek egymástól, vagyis úgy, hogy mindegyik csak a szálvezérlővel tartson kapcsolatot.



A kapcsolatok közül eddig még csak arról volt szó, hogy a vezérlőpanelnek ismernie kell a vezérlőt, hiszen neki kell szólnia, ha megnyomták valamelyik gombját. Azonban az ismeretségnek kölcsönösnek kell lennie, hiszen a vezérlőnek meg tudnia kell, hogy melyik panelt kérje meg az adatok kiíratására.
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 majd egy másik segédletben lesz szó. (*tovabbi_halas_variaciok.pdf*).

Az osztályok közötti kapcsolat az `AkvariumFrame`-n alakítható ki. Mivel beállításról van szó, a metódus most meghívható a frame konstruktorából (vagy a konstruktorból hívott `beallitas()` metódusból – természetesen nem muszáj külön metódust írni, csak most így talán jobban látszik, hogy mit kell beállítanunk):

```
private void beallit() {
    Vezerlo vezerlo = new Vezerlo();
    vezerlo.setRajzPanel(rajzPanel1);
    vezerloPanel1.setVezerlo(vezerlo);
    rajzPanel1.setVezerlo(vezerlo);
}
```

A rajzpanelnek nagyon egyszerű dolga van: ki kell rajzolnia azt, amit a vezérlő mond, illetve szólni kell neki, ha rákattintottak a felületére:

```
@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);
}

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

A hal példányok létrehozásakor már szó volt a `Hal` osztály konstruktoráról, de most részletesebben meg kellene tárgyalni ezt is és a `Vezerlo` osztályt is.
Mindkét osztályról csak a kommentezett kódrészleteket szeretném közölni, ezért most röviden beszéljük meg a feladataikat.

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.

A vezérlő feladatai:

- Egy adott hal vízbedobása: ekkor be kell állítani a hal szükséges adatait, elindítani a szálat, és hozzáadni a kirajzolandó halak listájához.
- A halak kirajzolása.
- A rajzpanel frissítésének kérése
- Annak vizsgálata, hogy mi történjen, ha a rajzfelületre kattintottak.

Lényegében ennyi, de még egy dolgot kiemelek: a rajzpanel átvételekor be kell állítanunk az akvárium méreteit is. Mivel az összes hal ugyanabban az akváriumban úszkál, ezért ezek a méretek statikusak.

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

```
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();
}

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();
}

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

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

A `Vezerlo` osztály ide vonatkozó része:

```
public class Vezerlo {

    private RajzPanel rajzPanel;
    private VezerloPanel vezerloPanel;

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

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

    private void beallitas() {
        Hal.setAkvariumMelyseg(rajzPanel.getHeight()-Global.FELSO_KEPMERET);
        Hal.setAkvariumSzelesseg(rajzPanel.getWidth());
        Hal.setVizMagassag((int) (rajzPanel.getHeight()*
                                        Global.KEP_MAGASSAG_ARANY));
        zene = new Zene();
    }

    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()));
        long ido = (long) (Math.random()*(Global.FELSO_IDO-Global.ALSO_IDO)
                                                + Global.ALSO_IDO);
        int lepeskoz = (Math.random() < 0.5)? 1 : -1;
        hal.beallitas(kepX, kepY, true, this, ido, lepeskoz);
        uszoHalak.add(hal);
        // akkor van szükség frissítésre, ha nem akarunk szálat indítani
        // frissit();
        hal.setUszik(true);
        hal.start();
    }

    public void rajzolas(Graphics g) {
        for (Hal hal : uszoHalak) {
            hal.rajzolas(g);
        }
    }

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

```java
public void talalatVizsgalat(int x, int y) {
    boolean eltalaltakEgyet = false;
    for (Hal hal : halak) {
        if(hal.eltalaltak(x,y)){
            // ekkor a szál is leáll
            hal.setAktiv(false);
            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();
}
```