Halas feladat – fájlbeviteli része

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. Ez azt jelenti, hogy létre is jönnek a hal példányok: mindnek lesz neve, egyértelműen megadható hozzájuk a bal- és jobboldali profilképük és a képméret.
A neveket az *adatok.txt* fájl tartalmazza, a képek az *src\kepek* mappában találhatóak a nevek sorrendjében, a méretük pedig véletlenül generálható a megadott határok között. Az adatfájlt egy fájlválasztó segítségével keressük meg. Adatbetöltés után megjelenik a nevük a listafelületen. A „Vízbedobás” gomb csak akkor válik aktívvá, ha sikerült beolvasni az adatokat.

Megoldásrészletek

Most csak a fájlból való adatbevitelről lesz szó.

1. Adatbevitel
   Ehhez szükség lesz a `Hal` osztály konstruktorára
   a képpárokat tartalmazó `KepPar` osztályra és
   az általánosan használt konstansokat tartalmazó `Global` osztályra

2. Vezérlőpanel
   az adatbevitel gomb hatása

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.

Az osztály konstruktora és a hozzá tartozó adattag-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 a halhoz tartozó kép-párokat tartalmazza:

```
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 kétféle módon is meg akarjuk írni (az adatbázisból való olvasás hf ☺), induljunk ki most is egy interfészből.

Ha csak szigorúan erre a feladatra koncentrálnánk, akkor akár azt is megtehetnénk, hogy a vezérlő-panelen írjuk meg a teljes beolvasást, vagy csak arra figyelnénk, hogy a beolvasott hal példányok-nak egy listamodellbe kell kerülniük, és ezért közvetlenül modellbe olvasnánk az adatokat. Ha azonban megpróbáljuk magunkévá tenni azt a szemléletet, amely szerint mindig gondolunk a könnyű módosíthatóságra, akkor célszerű ezt is függetleníteni a konkrét feladattól, vagyis az adatszerkezetet is és a vezérlést is próbáljuk minél általánosabban kezelni.

Ebből egyrészt az következik, hogy az adatokat egy `List` típusú eredményt visszaadó metódus hozza létre, vagyis gondolni kell arra, hogy esetleg konzolos felületen akarják használni ezeket a példányokat. Persze, a konzolos alkalmazásnak némileg ellentmond az, hogy képeket is akarunk használni ☺. Az viszont lehet érv a lista mellett, hogy esetleg egy másik panelen nem default-, hanem másfajta modellbe szeretnénk rakni őket. Úgyhogy úgy oldjuk meg, hogy a fájlbeolvasó osztályban listába olvassuk az adatokat.

Másrészt a vezérlést pedig úgy írjuk meg, hogy ha valakinek az jut eszébe, hogy mégsem a gomb-nyomás hatására olvassunk, hanem pl. a rajzpanelre kattintáskor, vagy egy újabb panelhez kapcsolt

esemény bekövetkeztekor, vagy esetleg a beolvasott halak listáját kellene máshol megjeleníteni, akkor se kelljen nagyon sokat változtatnunk. Ezért most úgy oldjuk meg, hogy a vezérlő osztályra bízzuk az olvasást, és a gombnyomás hatására a vezérlőpanel csak megkéri a vezérlőt, hogy olvassa be az adatokat.

Előbb az adatbeolvasó interfészt és osztályt mutatom meg:

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

Olvasáskor egy adatfájlból 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.
Vitatható az a megoldás, amelyet most választok, vagyis az, hogy a képfájlok elérési útvonalát tartalmazó stringbe a fájlnév közös részét is belevesszük, de most mégis így csináljuk. 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 pedig adott határok között véletlenszerűen adjuk meg.

```
public class FajlBevitel implements AdatInput{

    private File fajl;
    private String CHAR_SET = "UTF-8";
    private double felsoMeret;
    private double alsoMeret;
    private String kepfajlEleres;

    public FajlBevitel(File fajl, double felsoMeret, double alsoMeret,
                       String kepfajlEleres) {
        this.fajl = fajl;
        this.felsoMeret = felsoMeret;
        this.alsoMeret = alsoMeret;
        this.kepfajlEleres = kepfajlEleres;
    }
```

```
    @Override
    public List<Hal> halLista() throws Exception {
        List<Hal> halak = new ArrayList<>();
        try(Scanner fajlScanner = new Scanner(fajl, CHAR_SET)){
            int sorszam = 0;
            String nev;
            int kepSzelesseg, kepMagassag;
            Image balKep, jobbKep;
            KepPar kepPar;
            while (fajlScanner.hasNextLine()) {
                nev = fajlScanner.nextLine();
                kepSzelesseg =
                     (int) (Math.random()*(felsoMeret - alsoMeret) + alsoMeret);
                kepMagassag =
                     (int) (Math.random()*(felsoMeret - alsoMeret) + alsoMeret);
                balKep =
                    new ImageIcon(this.getClass().
                    getResource(kepfajlEleres + sorszam + "_BAL.png")).getImage();
                jobbKep =
                    new ImageIcon(this.getClass().
                    getResource(kepfajlEleres + sorszam + "_JOBB.png")).getImage();
                kepPar = new KepPar(balKep, jobbKep);
                halak.add(new Hal(nev, kepSzelesseg, kepMagassag, kepPar));
                sorszam++;
            }
        }
        return halak;
    }
}
```

Gyakorlaton kicsit felemás módon oldottuk meg a feladatot, ugyanis a vezérlőpanel döntötte el, hogy egy dialógusablakból kiválasztott fájlból olvassunk, vagyis ha adatbázisból szeretnénk, akkor mégiscsak hozzá kellene nyúlni a vezérlőpanel osztály kódjához is. Most még általánosabb megoldást mutatok, mégpedig olyat, hogy valóban csak a vezérlő osztály kódját kelljen kicsit módosítani, ha mégis adatbázisból szeretnénk olvasni.

Az adatbevitelt úgy oldjuk meg, hogy közben figyelünk az MVC/MVP szemléletre.
(A teljesen „tisztességes” változatot csak a *tovabbi_halas_variaciok.pdf* fájlban mutatom majd meg.)

A szemlélet szerint a panelek kizárólag megjelenítésre valók, vagyis úgy kellene megoldani a feladatot, hogy abban pillanatok alatt ki lehessen cserélni a paneleket, vagyis tényleg csak a megjelenítést és a vezérlő osztállyal való kapcsolatot tartalmazzák. Ezért nem itt, hanem a vezérlő osztályban írjuk meg az `adatBeolvasas()` metódust. Ez azonban további kérdéseket és módosítandó részleteket vet fel. Egyrészt, ha igazán általánosan szeretnénk megoldani a feladatot, akkor a fájlválasztást is a vezérlőre kellene bízni. A másik probléma az, hogy valahogy meg kell oldani, hogy a gombok aktivitása csak akkor változzon meg, ha sikeres az olvasás. A probléma megoldása az, hogy a vezérlőben `boolean` metódusként írjuk meg az adatbeolvasást, és a panel majd akkor hívja meg a gombváltást, ha sikeresen futott le a beolvasás.

A harmadik megoldandó részlet az, hogy most az adatok a vezérlő osztályban vannak, valahogyan át kellene kerülniük a panel listafelületére – ezt meg lehet oldani az `adatBeolvasas()` metóduson belül, itt kell átadni őket a vezérlőpanelnek. Mivel beolvasás után azonnal feldolgozzuk

a beolvasott lista adatait, ezért azt elég csak lokálisan deklarálni – a vezérlő osztálynak nincs szüksége a teljes listára, csak a vízben lubickolókra.

A negyedik: a listafelületre való kiíratás miatt a vezérlő osztálynak ismernie kell a vezérlő panelt, vagyis deklarálni kell egy ilyen típusú példányt, és lehetővé tenni, hogy értéket kapjon (vagy a konstruktorban kell szerepeltetni, vagy settert kell írni hozzá).

A vezérlőpanelen az adatbevitelért felelős gombnyomás a vezérlő `adatBeolvasas()` metódusát hívja meg. Ehhez deklarálni kell egy `Vezerlo` típusú `vezerlo` változót, és persze, a hozzá tartozó settert is meg kell írni. Miután a vezérlő beolvassa az adatokat, átadja őket a panelnek. Ezt dolgozza fel a panel a `listabaIr()` metódusban. A `VezerloPanel` beállításai és az adatbeolvasás- és megjelenítés lépései:

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

    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;
    private Vezerlo vezerlo;

    private DefaultListModel<Hal> halModell = new DefaultListModel<>();

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

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

```
    private void btnAdatbevitelActionPerformed(java.awt.event.ActionEvent evt) {
        if(vezerlo.adatBeolvasas()) gombAktivitas(true);
    }

    public void listabaIr(List<Hal> beolvasottHalak) {
        for (Hal hal : beolvasottHalak) {
            halModell.addElement(hal);
        }
    }
```

A `Vezerlo` osztály hivatkozott beolvasási metódusa:

```
    public boolean adatBeolvasas() {
        try {
            JFileChooser fajlValaszto = new JFileChooser(new File("."));
            if (fajlValaszto.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
                File fajl1 = fajlValaszto.getSelectedFile();
                AdatInput adatInput = new FajlBevitel(fajl1, Global.FELSO_KEPMERET,
                        Global.ALSO_KEPMERET, Global.KEPFAJL_ELERES);
                List<Hal> beolvasottHalak = adatInput.halLista();
                vezerloPanel.listabaIr(beolvasottHalak);
                return true;
            }
        } catch (Exception ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);

        }
        return false;
    }
```