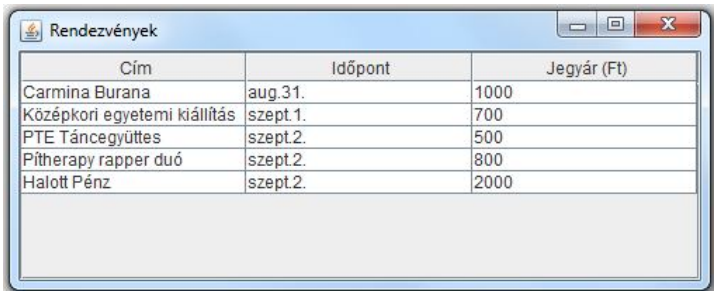
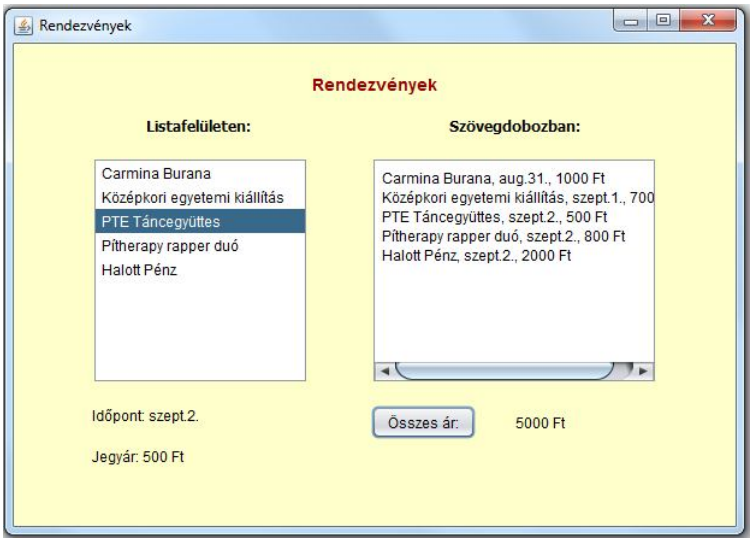
Bemelegítés

Bemelegítésként oldjunk meg két egyszerű feladatot!

1. Adott néhány rendezvény (címe, dátuma (most csak String) és a jegyár). Olvassuk be az adatokat, majd jelenítsük meg őket egy grafikus felületen:



2. Ugyanezeket a rendezvényeket beolvasva, most másik felületen jelenítsük meg:



Egyrészt írassuk ki egy listafelületre is, másrészt egy szövegdobozba (főleg azért, hogy lássa a kettő közötti különbséget). A listafelület egy elemére kattintva, a lista alatt jelenjen meg a kiválasztott rendezvény időpontja és jegyára. Az „Összes ár” feliratú gombot megnyomva jelenjen meg a rendezvények jegyárainak összege.

Megoldásrészletek:

1. A feladat konzolos megoldása ez, vagy ehhez hasonló lehetne:

```
public class Rendezveny {

    private String cim;
    private String idoPont;
    private int jegyAr;

    public Rendezveny(String cim, String idoPont, int jegyAr) {
        this.cim = cim;
        this.idoPont = idoPont;
        this.jegyAr = jegyAr;
    }

    @Override
    public String toString() {
        return cim + ", időpontja: " + idoPont + ", jegyár: " + jegyAr + " Ft";
    }
```

+ getterek, setterek.

Eddig általában úgy oldottuk meg a vezérlést, hogy a `main()` metódust tartalmazó `Main` osztályt példányosítottuk, és meghívtuk annak `start()` metódusát, ami gyakorlatilag elvégezte a vezérlést. Lényegében most is ugyanezt tesszük, csak a későbbiek kedvéért nem a `Main` osztályt példányosítjuk, hanem létrehozunk egy külön `Vezerles` osztályt, ami ugyanazt csinálja, mint korábban a `Main` osztály metódusai:

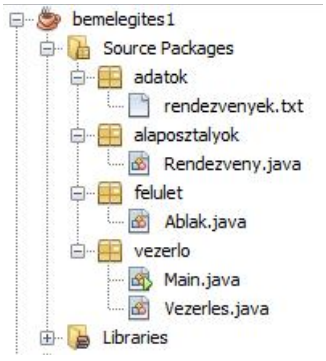
```
public class Main {

    public static void main(String[] args) {
        new Vezerles().start();
    }
}

public class Vezerles {

    private final String RENDEZVENY_ELERES = "/adatok/rendezvenyek.txt";
    private final String CHAR_SET = "UTF-8";

    private List<Rendezveny> rendezvenyek = new ArrayList<>();

    public Vezerles() {
    }

    void start() {
        adatBevitel();
        adatKiiras();
        ablakba();
    }
```

Az adatbevitel, adatkiírás ugyanaz, mint eddig, az `ablakba()` metódus hatására kerülnek majd az adatok grafikus felületre is.

```
private void adatBevitel() {

    try (InputStream ins = this.getClass().getResourceAsStream(RENDEZVENY_ELERES);
            Scanner fajlScanner = new Scanner(ins, CHAR_SET)) {

        String cim, idoPont;
        int ar;
        String sor, adatok[];
        Rendezveny rendezveny;
        while (fajlScanner.hasNextLine()) {
            sor = fajlScanner.nextLine();
            adatok = sor.split(";");
            cim = adatok[0];
            idoPont = adatok[1];
            ar = Integer.parseInt(adatok[2]);
            rendezveny = new Rendezveny(cim, idoPont, ar);
            rendezvenyek.add(rendezveny);
        }
    } catch (IOException ex) {
        Logger.getLogger(Vezerles.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void adatKiiras() {
    System.out.println("A rendezvények: ");
    for (Rendezveny rendezveny : rendezvenyek) {
        System.out.println(rendezveny);
    }
}
```

Az ablakba íráshoz előbb meg kell beszélnünk, hogy egyáltalán hogyan is lehet ablakot létrehozni. Ez nem jelent túl nagy gondot, hiszen a Java Swing csomagjában létezik `JFrame` osztály, csak kicsit testre kell szabnunk. Pontosan ezt a testre szabást végzi az öröklődés, vagyis létre kell hoznunk egy olyan osztályt, amely kiterjeszti a `JFrame` osztályt:

```
public class Ablak extends JFrame{
```



A projekt lehetséges szerkezete:

Be kell állítanunk az ablak méreteit, esetleg címet kell adnunk neki, középre igazíthatjuk, stb., és ami nagyon fontos: láthatóvá kell tennünk. Ezt megoldhatjuk úgy, hogy a vezérlésben állítjuk be ezeket az értékeket és konstruktoron keresztül adjuk át az `Ablak` osztálynak, de úgy is megoldható, hogy az `Ablak` osztályon belül állítjuk be őket. A táblázat létrehozása és adatokkal való feltöltése

ennek az `Ablak` osztálynak a dolga. A vezérlés `ablakba()` metódusa:

```
    private void ablakba() {
        int szelesseg = 500, magassag = 200;
        String cim = "Rendezvények";
        Ablak ablak = new Ablak(szelesseg, magassag, cim);
        String[] oszlopNevek = {"Cím","Időpont","Jegyár (Ft)"};
        ablak.ablakbaIr(new ArrayList<Rendezveny>(rendezvenyek),oszlopNevek);
    }
```

Az `Ablak` osztály kódja – majd néhány magyarázó mondat lesz még utána:

```
public class Ablak extends JFrame{
    private int szelesseg;
    private int magassag;
    private String cim;

    public Ablak(int szelesseg, int magassag, String cim) {
        this.szelesseg = szelesseg;
        this.magassag = magassag;
        this.cim = cim;
        inicializalas();
    }

    private void inicializalas() {
        this.setSize(szelesseg, magassag);
        this.setTitle(cim);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public void ablakbaIr(List<Rendezveny> rendezvenyek, String[] oszlopNevek) {
        //  oszlopNevek = {"Cím","Időpont","Jegyár (Ft)"};
        String[][] adatok = new String[rendezvenyek.size()][oszlopNevek.length];
        for (int i = 0; i < rendezvenyek.size(); i++) {
            adatok[i][0] = rendezvenyek.get(i).getCim();
            adatok[i][1] = rendezvenyek.get(i).getIdoPont();
            adatok[i][2] = String.valueOf(rendezvenyek.get(i).getJegyAr());
        }

        TableModel tableModel = new DefaultTableModel(adatok, oszlopNevek);
        JTable tabla = new JTable(tableModel);
        JScrollPane jScrollPane = new JScrollPane(tabla);

        this.add(jScrollPane);
        this.revalidate();
    }
}
```

Az inicializáláskor a méreten, címen, középre igazításon kívül azt is meg kell mondani, hogy mi történjen, ha az ablak bezáró gombjára kattintunk, és ahogy már szó volt róla, valahol muszáj láthatóvá tenni.

A táblázat létrehozásának nem ez az egyetlen módja, de ez viszonylag egyszerű: a kiírandó adatokból egy string tömböt hozunk létre, majd ebből és az oszlopneveket tartalmazó tömbből létrehozunk egy tábla-modellt. Mi is ez a modell? Majd később kicsit részletesebben is lesz szó róla, de a Swing a közismert és közkedvelt MVC (model – view - controller) tervezési minta alapján készült. Ez azt jelenti, hogy amennyire csak lehet, különválasztjuk az adatokat (modell), a felületet (view) és a vezérlést (controller). A Swing ezt az elvet követi, vagyis minden komponens-felülethez tartozik egy, az adatokat tartalmazó modell (egy modellhez akár több felületet is rendelhetünk, illetve egy felület modellje is cserélődhet a futás során). A komponensekhez események rendelhetők (eseményfigyelők (listener) figyelik, hogy bekövetkezett-e az adott esemény). Ha bekövetkezett az esemény, akkor annak hatására történik valami – ez a vezérlés.

Esetünkben a `JTable` felülethez egy `TableModel` példány tartozik. Ezt a táblafelületet egy `JScrollPane` típusú mezőre rakjuk. Ez azért kell, mert ha a táblázatba több adat kerül, mint amennyi olvasható a megadott felületen, akkor automatikusan megjelenik a szükséges gördítő sáv (scroll).

A `revalidate()` metódus frissíti a felületet. Ezt el is lehetett volna hagyni, ha csak itt tesszük láthatóvá az ablakot.

Láthatjuk tehát, hogy konzolos alkalmazásból is létre tudunk hozni grafikus felületet. Sejthető azonban, hogy kicsit is összetettebb felületek esetén ez nem túl kényelmes megoldás.

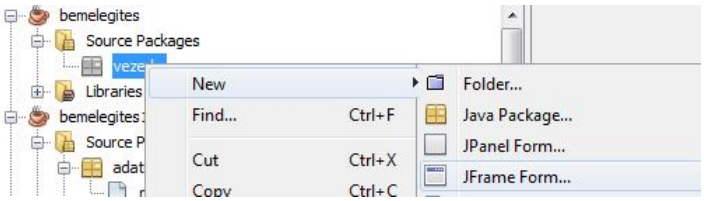
Szerencsére a fejlesztő környezetek sok segítséget nyújtanak. A második bemelegítő feladatot már így oldjuk meg.

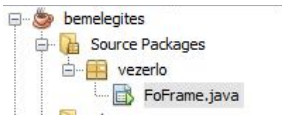
2. Először is hozzuk létre a szükséges felületet. Nagyon röviden ismertetjük azt, hogyan hozható létre a NetBeans segítségével, de ha eddig más fejlesztő környezetet használt, akkor továbbra is maradhat annál, csak kell egy kis önálló munka ahhoz, hogy utánanézzen, az a környezet milyen módon segíti a grafikus felület létrehozását.

Mint az előző megoldásban láttuk, szükségünk van egy `JFrame` típusú osztályra. Ez generálható is.

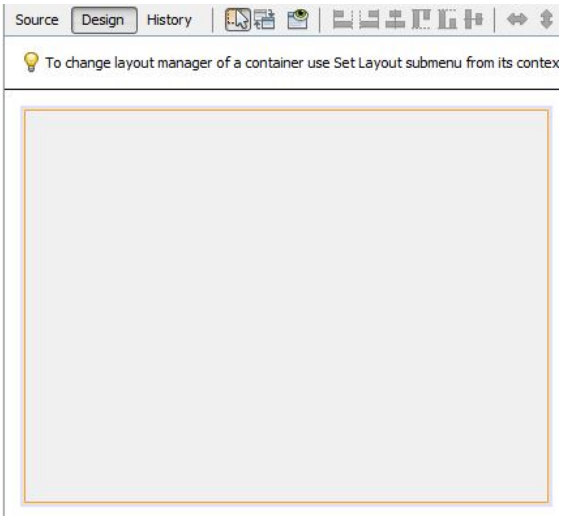
Egy vadonatúj projekt létrehozásakor, most ne generáltasson külön `Main` osztályt, mert a generált `JFrame` osztály tartalmazza a `main()` metódust. De most nem muszáj új projektet létrehozni, módosíthatjuk az előzőt – vagy még jobb, ha létrehozunk egy üres új projektet és a korábbi projekt csomagjait átmásoljuk bele. A másolásból kihagyhatjuk a `Main` osztályt (vagy utólag is ki lehet törölni).

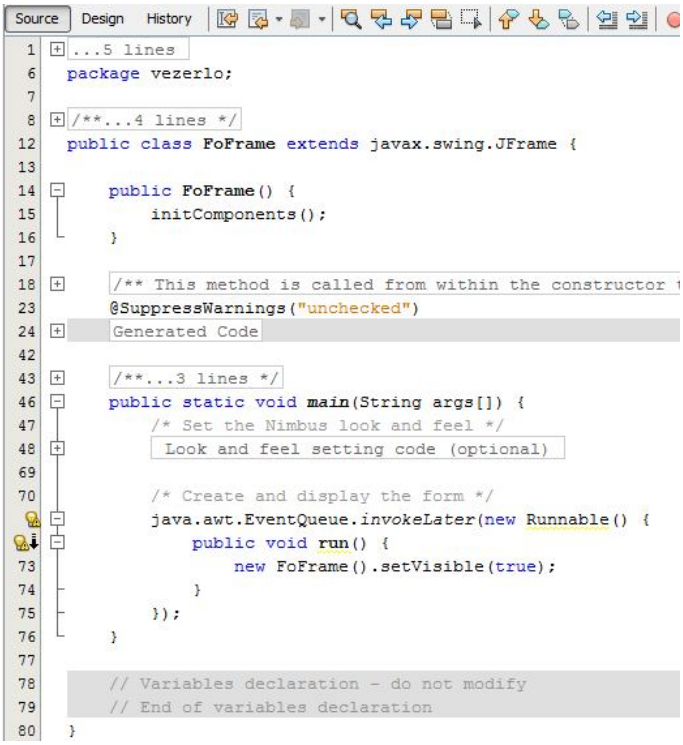
A vezerlo csomagban hozzunk létre egy új `JFrame` form-ot.





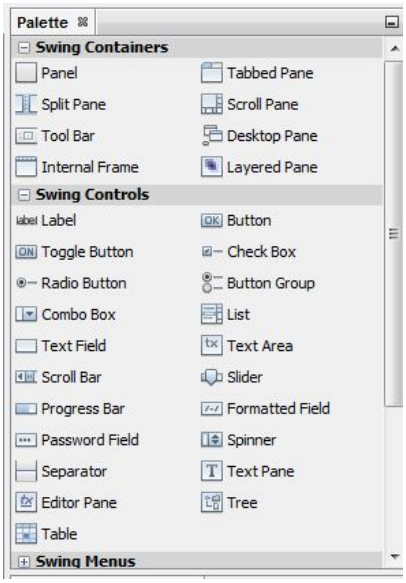
A `FoFrame` osztályt két „nézetből” szerkeszthetjük:





Az egyik a „Design” – itt tervezhetjük meg a felületet, a másik a „Source”, ami a kód leírására szolgál.

Figyelje majd meg a projekt mappájában, hogy a külalak egy xml fájl segítségével van leírva. Ugyancsak figyelje meg a generált kódot is, főleg akkor, amikor már gazdagabb a felület. Látni fogja, hogy itt van leírva az inicializálás, azaz az `initComponents()` metódus.



Design módban láthatja a palettát is, innen tudja ráhúzni a felületre a szükséges komponenseket.

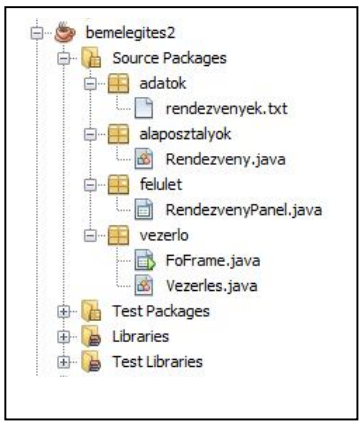
Ezt akár ki is próbálhatja, és húzzon rá a frame felületre mondjuk egy gombot, és futtassa. De próbaként ennyi elég is, mert nem így csináljuk, így legföljebb csak játszadozni lehet.

A frame szó keretet jelent, és valóban ez is az osztály funkciója: keretet ad a feladatmegoldáshoz.

Előbb egy (vagy több) panelt rakunk fel erre a frame felületre, és csak a panelre jön a többi. A panel fogja össze az összetartozó komponenseket.

De még ne kapkodja el, a panelt nem a palettáról húzzuk rá a frame-re. Mégpedig azért nem, mert az lényegében beégetésnek számít: bajban lennénk, ha ki kellene cserélni, vagy ha esetleg utólag jutna a megrendelő eszébe, hogy több kartotékfület szeretne – ekkor, fixen rárakott panel esetén dobhatjuk ki az eddigi munkánkat. Ha viszont saját panel osztályt írunk, akkor pillanatok alatt megoldható a csere is, módosítás is.

Mindezt végiggondolva a projekt szerkezete:



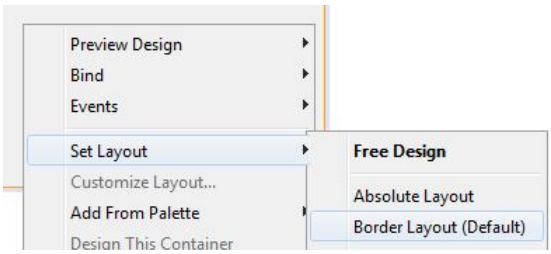
A `RendezvenyPanel` a frame-hez hasonló módon generálható, csak ekkor egy új JPanel form-ot hozunk létre. Ennek is lesz egy design- és egy source-módja.

Egy ilyen felületet hozunk létre – ehhez már használjuk a palettán felkínált elemeket, amelyeket ráhúzunk a panel felületére:

A komponensen jobb egérgombot nyomva megnyithatjuk a Properties ablakot is, amelyben be tudjuk állítani a komponensek tulajdonságait, pl. a feliratát (text), a ráírt betűtípust, középre tudjuk igazítani a szöveget, színt tudunk változtatni, stb. Nagyon fontos, hogy azokat a komponenseket, amelyekre kódból is hivatkozunk, át is nevezzük (jobb egérgomb change variable name). A névadás legyen mindig beszédes, vagyis utaljon rá, hogy milyen típusú komponensről van szó, és arra is, hogy mi a funkciója. Nem kötelező, de szokásos megoldás, hogy a komponens típusa kerül a név elejére, mégpedig magánhangzók nélkül (pl. Label – lbl, List – lst, Button – btn, Text Area – txt, stb.), és a név második része utal a funkcióra. Amelyikre nem hivatkozunk, ott maradhat az eredetileg generálódott név.

A következő ábra a megoldás során használt elnevezéseket mutatja. A szürkével jelzett részek az elrendezés jobb kialakítását jelzik, de ha valakit zavar, ki is lehet kapcsolni:

Ha kialakítottuk a felületet, akkor a saját panel osztályból készült példány ugyanúgy ráhúzható a frame felületére, mint a készen kapott „gyári” példány. Ehhez az kell, hogy előbb fordítsuk le a projektet. Ha hibátlanul lefordult, akkor a projektből a RendezvenyPanel.java panelt kell ráhúzni a frame design felületére. A frame felületének elrendezését előtte Border Layout típusúra kell állítani: felület – jobb egérgomb, és:



Ekkor a panelt a frame méretéhez igazítja.

A frame méretét a kódban kell megadnunk, ugyanis a használt layout-tól függ, hogy figyelembe veszi-e a tervezési méretet. A panel design módjában azonban célszerű beállítani a helyes méretet (a panel alsó vagy jobboldali keretére duplán kattintva lehet megadni). Sőt, ha nagyon precízen szeretnénk dolgozni, akkor tervezéskor azt is figyelembe vehetjük, hogy a panel a frame belső felületére kerül, vagyis a belső mérethez hozzáadhatjuk a frame keretének méreteit is: bal-, alsó- és jobb-keret 8 pixel, felső 30 pixel.

Ahhoz, hogy a saját panelt rá lehessen húzni a frame felületére, egyrészt az kell, hogy helyesen forduljon le a projekt, másrészt az, hogy a panel úgynevezett Java bean legyen, vagyis legyen paraméter nélküli konstruktora. Ez utóbbi alapértelmezetten ilyen, csak „elrontani” lehet, de ez nyilván nem lehet célunk.

Mivel le kell fordulnia a projektnek, célszerű minél hamarabb rárakni a panelt a frame-re, vagyis akkor, amikor még gyakorlatilag nem is írtunk saját kódot, hanem csak a felület van készen. Ha utólag jut eszünkbe még újabb komponenst felrakni a panelre, akkor ez nem okoz gondot, futtatáskor már a módosított felület látszik, egyetlen esettől eltekintve: Ha üres panelt rakunk a frame-re, majd utólag próbáljuk módosítani a panel felületét, akkor futáskor nem látjuk a módosítást. Ha azonban van már rajta legalább egy komponens, amikor felrakjuk a frame-re, akkor ezek után utólag már észleli a módosításokat.

Ha netalántán nem sikerül ráhúzni a panelt a frame-re, akkor használja a frame `add()` metódusát, és így adja hozzá a panel egy példányát.

Ha létrehozta a felületet, nézze meg a panel generált kódját is, sőt, esetleg a létrejött xml fájlt is, hogy lássa, a felületkialakítás természetesen kódolást von maga után.

A felület kialakítása után lássuk a kódot. Ha elég ügyesek vagyunk, akkor alig kell módosítani valamit az eredetin. A `Rendezveny` osztály és az `adatBevitel()` metódus szó szerint ugyanaz. Már nyilván nincs szükségünk a konzolos kiíratásra, helyette a panelre írjuk az adatokat. Ami kérdéses: hol hívjuk meg ezeket a metódusokat. Az eredeti változatban a `main()` metódus példányosította a `Vezerles` osztályt, és meghívta annak `start()` metódusát, ami elindította a beolvasást és kiíratást. Ugyanezt – egy kis módosítással – most is megtehetjük. A `main()` metódus most a frame osztályban van. Az indításon kívül még a láthatóvá tétel is feladata, vagyis ezt is meg kell hívnunk a metódusban. A vezérlés most nem konzolra, hanem a panelre írja az adatokat, vagyis a panelt is ismernie kell, mégpedig ugyanazt, amelyet ráhúztunk a frame-re. Ezért úgy kell példányosítanunk a vezérlést, hogy az a panel példányt is megkapja. Ennek legegyszerűbb módja, ha a panelt a vezérlés konstruktorában adjuk át, de természetesen lehetne setterrel is.

A frame konstruktora és a futást indító metódus:

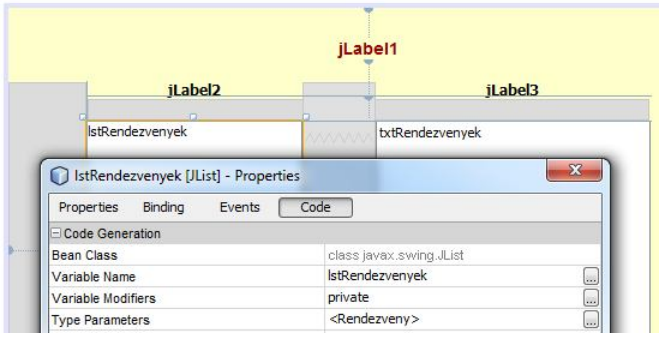
```
public class FoFrame extends javax.swing.JFrame {

    private final int SZELESSEG = 616;
    private final int MAGASSAG = 438;
    private final String CIM = "Rendezvények";

    public FoFrame() {
        initComponents();
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(CIM);
        this.setLocationRelativeTo(null);
    }


    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().start();
            }
        });
    }

    private void start() {
        this.setVisible(true);
        Vezerles vezerles = new Vezerles(rendezvenyPanel1);
        vezerles.start();
    }
```

A `Vezerles` osztály:

```
public class Vezerles {

    private final String RENDEZVENY_ELERES = "/adatok/rendezvenyek.txt";
    private final String CHAR_SET = "UTF-8";

    private List<Rendezveny> rendezvenyek = new ArrayList<>();
    private RendezvenyPanel rendezvenyPanel;

    public Vezerles(RendezvenyPanel rendezvenyPanel) {
        this.rendezvenyPanel = rendezvenyPanel;
    }

    void start() {
        adatBevitel();
        adatKiiras();
    }

    private void adatBevitel() {

        try (InputStream ins = this.getClass().getResourceAsStream(RENDEZVENY_ELERES);
                Scanner fajlScanner = new Scanner(ins, CHAR_SET)) {

            String cim, idoPont;
            int ar;
            String sor, adatok[];
            Rendezveny rendezveny;
            while (fajlScanner.hasNextLine()) {
                sor = fajlScanner.nextLine();
                adatok = sor.split(";");
                cim = adatok[0];
                idoPont = adatok[1];
                ar = Integer.parseInt(adatok[2]);
                rendezveny = new Rendezveny(cim, idoPont, ar);
                rendezvenyek.add(rendezveny);
            }
        } catch (IOException ex) {
            Logger.getLogger(Vezerles.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void adatKiiras() {
        rendezvenyPanel.listaFeluletreIr(rendezvenyek);
        rendezvenyPanel.szovegDobozbaIr(rendezvenyek);
    }
}
```

Beszéljük meg a rendezvénypanel két kiíró metódusát. A szövegdobozba írás roppant egyszerű, csak hozzá kell fűzni az új kiírandó sort a többihez. A listakezelés kicsit összetettebb. Ahogy a táblázat létrehozásakor már volt szó róla, a Swing komponensek az

MVC tervezési mintát próbálják követni (és ugyanezt a modellt követi az is, hogy a projektünkben meghagytuk a vezérlő osztályt, de erre majd kicsit később még visszatérünk). A minta szerint különválasztjuk a felületet (view), az adatokat (model) és a vezérlést (controller). Ezt a felépítést követi a listakezelés is. A `JList` példány a listafelületet jelenti, amelyhez hozzá kell majd rendelnünk a megjelenítendő adatokat tartalmazó listamodellt. Ez jelenleg egy `DefaultListModel` típusú példány (a default jelzőből sejthető, hogy lehet másmilyen modell is). Itt is generikus módon deklaráljuk ezt a példányt, vagyis előírjuk, hogy milyen típusú példányok lehetnek az adott modellben. Ugyanezt a `JList` példány esetén is meg kell mondanunk. Ezt a panel Design módjában tehetjük meg: listafelület, jobb egérgomb, Properties, itt a Code fülön tudjuk beállítani a Type Parameters értékét (alapértelmezetten <String> – ezt kell átírnunk).



A panel eddig megbeszélt kódrészlete:

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

    private DefaultListModel<Rendezveny> rendezvenyModel
                                            = new DefaultListModel<>();

    public RendezvenyPanel() {
        initComponents();
        lstRendezvenyek.setModel(rendezvenyModel);
    }

    public void listaFeluletreIr(List<Rendezveny> rendezvenyek) {
        for (Rendezveny rendezveny : rendezvenyek) {
            rendezvenyModel.addElement(rendezveny);
        }
    }

    public void szovegDobozbaIr(List<Rendezveny> rendezvenyek) {
        for (Rendezveny rendezveny : rendezvenyek) {
            txtRendezvenyek.append(rendezveny.szovegDobozba()+"\n");
        }
    }
```

Hátra van még az események kezelése. Grafikus komponensek esetén ezek jelentik a vezérlést. Természetesen mi magunk is megírhatjuk az eseményfigyelők kódját, de egyszerűbb, ha ezeket is generáljuk. Ezt szintén a Design módban tehetjük meg: a komponensen jobb egérgombot nyomva, az Events menüpontból kiválasztható a figyelendő esemény, és generálódik az esemény bekövetkeztekor végrehajtandó metódus váza is, csak meg kell írnunk a tartalmát.

Az egyik esemény az, hogy a listafelület egy elemére kattintva a listafelület alatt megjelennek a kért adatok. Most tehát a listafelülethez kell eseményt rendelnünk. Előtte azonban állítsuk be azt, hogy egyszerre csak egy elemet lehessen kiválasztani a felületről: jobb egérgomb, Properties, selectionMode – és ezt állítsuk SINGLE tulajdonságúra.

A listafelülethez alapértelmezetten a ListSelection menüpont valueChanged eseményét szokás hozzárendelni. Ez akkor következik be, ha valami változás történik a listafelületen. Időnként azonban az is előfordulhat, hogy nem a felületen történik változás, hanem a kiválasztott elem valamilyen tulajdonsága változik, és pont erre a tulajdonságra vagyunk kíváncsiak akár többször is egymás után. Ekkor nem ezt az eseményt választjuk, hanem az egérkattintást, de ennek az a veszélye, hogy a kattintást akkor is érzékeli, ha pl. valami miatt inaktív a listafelület, vagyis ennek az eseménynek a precíz kezelése komoly körültekintést igényel. De esetünkben maradunk a valueChanged esemény kezelésénél. Az ehhez tartozó metódus:

```
    private void lstRendezvenyekValueChanged(javax.swing.event.ListSelectionEvent evt) {
        Rendezveny kivalasztott = lstRendezvenyek.getSelectedValue();
        lblIdopont.setText("Időpont: " + kivalasztott.getIdoPont());
        lblJegyAr.setText("Jegyár: " + kivalasztott.getJegyAr() + " Ft");
    }
```

A gombhoz az `actionPerformed` esemény tartozik. Ezt elvileg megoldhatnánk így:

```
    private void btnOsszArActionPerformed(java.awt.event.ActionEvent evt) {
        int ossz = 0;
        for (int i = 0; i < rendezvenyModel.size(); i++) {
            ossz += rendezvenyModel.get(i).getJegyAr();
        }
        lblOsszAr.setText(ossz + " Ft");
    }
```

Jól is működne, és kisebb, egyszerűbb feladatoknál szokták is így csinálni. Vagyis úgy, hogy kissé megsértjük a saját projektünk MVC szemléletét, és a panelhez nem csak megjelenítési funkciókat rendelünk (ahogy a szemlélet kívánná), hanem kisebb vezérlési funkciót is, esetünkben pl. itt számolunk ki egy összeget.

Ha nagyon szigorúan vennénk a projekt MVC szemléletét, akkor valahogy így kellene megoldanunk ezt a részfeladatot:

A `RendezvenyPanel` osztályban:

```
private Vezerles vezerles;

public void setVezerles(Vezerles vezerles) {
    this.vezerles = vezerles;
}

private void btnOsszArActionPerformed(java.awt.event.ActionEvent evt) {
    List<Rendezveny> rendezvenyek
                          = Collections.list(rendezvenyModel.elements());
    int ossz = vezerles.osszJegyAr(rendezvenyek);
    lblOsszAr.setText(ossz + " Ft");
}
```

A `Vezerles` osztályban:

```
public int osszJegyAr(List<Rendezveny> rendezvenyek) {
    int ossz = 0;
    for (Rendezveny rendezveny : rendezvenyek) {
        ossz += rendezveny.getJegyAr();
    }
    return ossz;
}
```

Végül, mivel kiderült, hogy a `RendezvenyPanel` osztálynak is ismernie kell a `Vezerles` osztály aktuális példányát, ezért a frame `start()` metódusában a panel-példány számára is át kell adni a `vezerles` példányt:

```
private void start() {
    this.setVisible(true);
    Vezerles vezerles = new Vezerles(rendezvenyPanel1);
    rendezvenyPanel1.setVezerles(vezerles);
    vezerles.start();
}
```

Szemmel láthatóan jóval bonyolultabb megoldást kaptunk, mint az eredeti változat volt (az, ahol a gombnyomás eseményhez tartozó összeget magában a panel osztályban számoltuk). Sőt, tovább egyszerűsödne a megoldás, ha akár az egész `Vezerles` osztályt kihagynánk, és az ottani metódusokat is közvetlenül a panel osztályában írnánk meg. Ekkor nem lenne szükség a vezérlés `start()` metódusára, hanem a beolvasást és az adatok kiíratását a panel betöltésekor hívhatnánk meg. (panel: Ancestor – ancestorAdded esemény). Ha most az elsődleges célja az, hogy örülhessen egy egyszerűen előállított, működő programnak, akkor nyugodtan csinálhatja így, de a továbbiakban egyre inkább próbálunk az MVC szemléletmód felé közelíteni. Hogy ez miért jó, ha így – legalábbis első látásra – bonyolultabb módon tudjuk

megoldani a feladatot? Az ilyen pici feladatoknál valóban bonyolultabbnak tűnik, de kicsit is komolyabb programok esetén nagyon is célszerű szétválasztani egymástól az adatokat, a vezérlést és a megjelenítést. Egy jól strukturált programot könnyen át lehet alakítani úgy, hogy a változatlan programlogika eredményét más-más felületen láthassuk, akár úgy, hogy egyik alkalmazásban konzolos módon, másikban grafikus felületen, de úgy is, hogy más-más grafikus felületen. Az MVC szemléletmód teszi lehetővé, hogy könnyen cserélhessük a megjelenítési felületeket. Ehhez az kell, hogy a felület lehetőleg minél függetlenebb legyen a programlogikától.