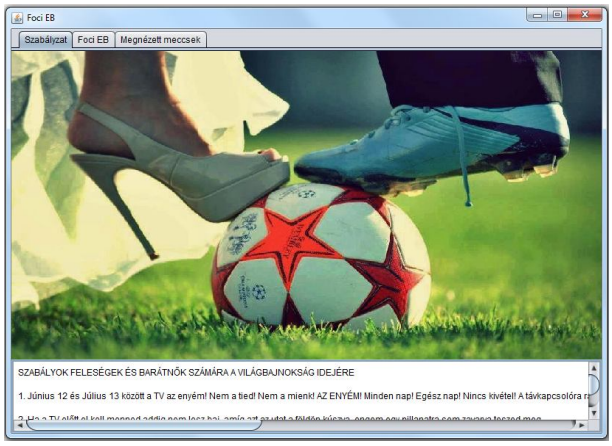
Foci EB 3.

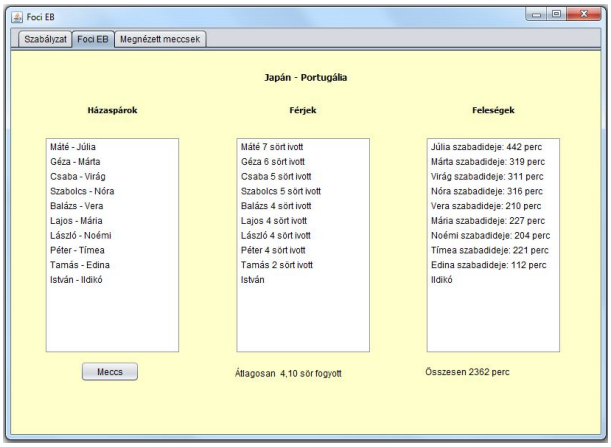
Folytassuk a korábbi feladatot, és bővítsük ki a felületet két további kartotékfüllel és a hozzájuk tartozó felületekkel:

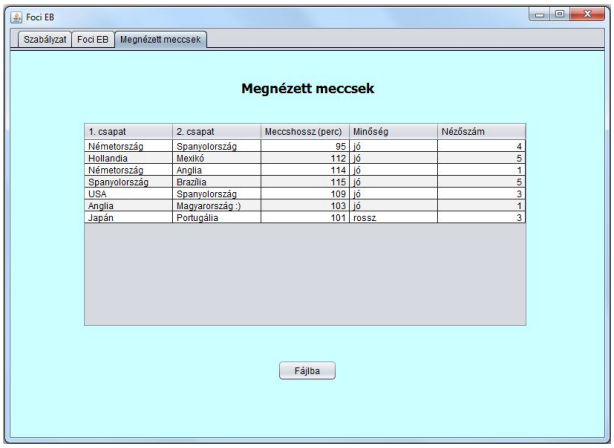
Az első felületen jelenjen meg a mellékelt kép, alatta pedig lehessen olvasni a szabályzatot (amely még a braziliai VB alkalmából született).

A második felület az, amit már korábban létrehoztunk, a harmadikon pedig egy táblázatba foglalva lehessen látni, hogy eddig milyen meccsek voltak (a két csapat, a meccs időtartama, minősítése (jó vagy rossz) és a meccset néző házaspárok száma).

A „Fájlba” feliratú gomb hatására jelenjen meg egy fájlválasztó ablak, és az ott megadott szövegfájlba írjuk ki a meccsek adatait.



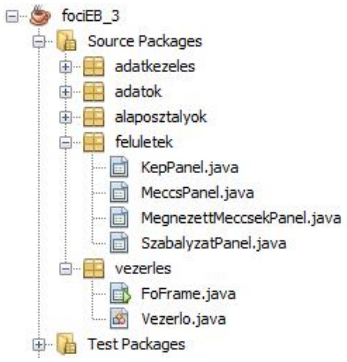




Megoldásrészletek:

Kezdjük a felület kialakításával. A frame-ről szedje le a rajta lévő panelt (a kódban pedig ideiglenesen kommentezze ki a rá vonatkozó kódrészleteket). A frame felületére most húzzon rá a palettáról egy TabbedPane (Swing Containers) típusú komponenst. Látszólag nem történik semmi, de valójában most alapoztuk meg azt, hogy több felület között is tudjunk majd válogatni a kartotékfülek segítségével. Erre rakjuk majd fel a különböző paneleket.

Hozzuk létre őket. A megoldás során a mellékelt felépítést és elnevezést használjuk majd. Azonnal felmerül a kérdés, hogy miért van négy panel, amikor csak három eltérő felületet szeretnénk létrehozni? Nyilván elég lenne három, de most azért oldjuk meg így, mert ha külön panelre rakjuk a képet, akkor azt könnyen tudjuk módosítani, másrészt az is cél, hogy lásson arra is példát, hogy egy panel tetejére is kerülhet másik panel.



Először hozzuk létre a `SzabalyzatPanel` felületét! Erre kerül rá egy `KepPanel` példány és egy szövegdoboz, amelybe majd a szabályzatot írjuk.

`KepPanel`:

Minden grafikus komponens örökli a `paintComponent()` metódust a `JComponent` osztályból (így az öröklött metódusok beszúrásának segítségével is generálható). Ebben a metódusban rajzoljuk meg a képet:

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

    private Image kep
        = new ImageIcon(this.getClass().getResource("/adatok/foci.jpg")).getImage();

    public KepPanel() {
        initComponents();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int szelesseg = this.getWidth(), magassag = this.getHeight();
        g.drawImage(kep, 0, 0, szelesseg, magassag, this);
    }
```

A képrajzoló metódus paraméterei: a kirajzolandó kép, a kép bal felső pontjának koordinátái, szélessége és magassága, illetve az utolsó paraméter egy `ImageObserver` típusú példány, amire ténylegesen csak akkor van szükség, ha aszinkron módon szeretnénk frissíteni a képet (pl. animált gif-et szeretnénk kirajzoltatni), de egyszerűbb képek esetén el is tekinthetünk tőle,

vagyis a paraméter `null` érték is lehet (a későbbiek során többször alkalmazzuk majd ezt a megoldást). Esetünkben a `this` a panel példányt jelenti, hiszen a `JPanel` osztály implementálja az `ImageObserver` interfészt.

A panel tervezési mérete most lényegtelen, hiszen a megadott méretű `SzabalyzatPanel` példány felületének méretéhez igazodik majd.

`SzabalyzatPanel`:

Ennek tervezési mérete a már ismert 800 × 500 pixel. A felület elrendezését állítsuk Border Layout-ra, és az alsó részére húzzon rá a palettáról egy TextArea példányt (`txtSzabalyzat` néven hivatkozunk majd rá). Figyeljen rá, hogy ráhúzáskor úgy engedje el, hogy valóban csak a felület alsó részét foglalja el – ügyetlen mozdulat esetén ugyanis előfordulhat, hogy a teljes felületet betakarja. Ha az ily módon létrejött szövegdoboz nem elég magas, akkor a tulajdonságok (Properties) között (vagy kódból) ezt megváltoztathatja. (preferredSize tulajdonság – esetleg a maximumSize értékét is át kell állítani). Figyelje meg, hogy a szövegdoboz felrakásakor automatikusan alákerül egy `JScrollPane` típusú példány, ez teszi lehetővé, hogy szükség esetén megjelenjenek a gördítő sávok. Ha módosítani akarja a méretet, akkor ennek a méreteit állítsa át. Mivel ezt a mezőt teljesen takarja a szövegdoboz, ezért ennek tulajdonság-paneljét a navigációs panelről tudjuk elérni, ott a `jScrollPane1` példány nevén jobb egérgombot nyomva.

Bár nem tartozik szorosan ide, de érdemes megjegyezni, hogy sok esetben kerül fel automatikusan a felületre egy-egy `JScrollPane` példány, amire nem árt odafigyelni. Ha például láthatatlanná akarjuk tenni ezt a szövegdobozt, akkor nem a szövegdoboz láthatóságát kell beállítanunk (ekkor látható marad az alatta lévő scroll-mező), hanem a `jScrollPane1` példányét.

Miután ráhúztuk a panel felületére a szövegdobozt (és szükség esetén átállítottuk a méretét), húzzuk fölé a saját `KepPanel` példányt. (Természetesen ezt csak akkor lehet, ha előtte lefordítottuk a projektet.) Látni fogja, hogy a kép kitölti a rendelkezésre álló helyet.

Mivel ez a felület inkább csak egy kis játék, ezért elvileg megoldhatnánk úgy is, hogy a panel betöltésének (ancestorAdded) eseményéhez rendeljük a fájlból való olvasást, és a beolvasott sorokat azonnal megjelenítjük a szövegdobozban. Ez a megoldás egyszerűnek tűnik, csak az a baj, hogy ha többször is visszanézünk erre az oldalra, akkor ismét és ismét betölti az adatokat és egyre többször jeleníti meg őket. Persze, mindezt kivédhetnénk pl. egy logikai változó bevezetésével, amely figyeli, hogy már be vannak-e olvasva az adatok, de talán ezt is szerencsésebb inkább a vezérlés hatóköre alá vonni.

Most az első változat bemutatásánál maradunk, de próbálja meg önállóan megoldani, hogy a vezérlés végzi a beolvasást, és ő kéri meg a panelt arra, hogy írja ki a szabályzat szövegét. Ha megértette az előző feladat megoldását, akkor ez könnyű feladat lesz.

A `SzabalyzatPanel` osztály a generált kódok nélkül:

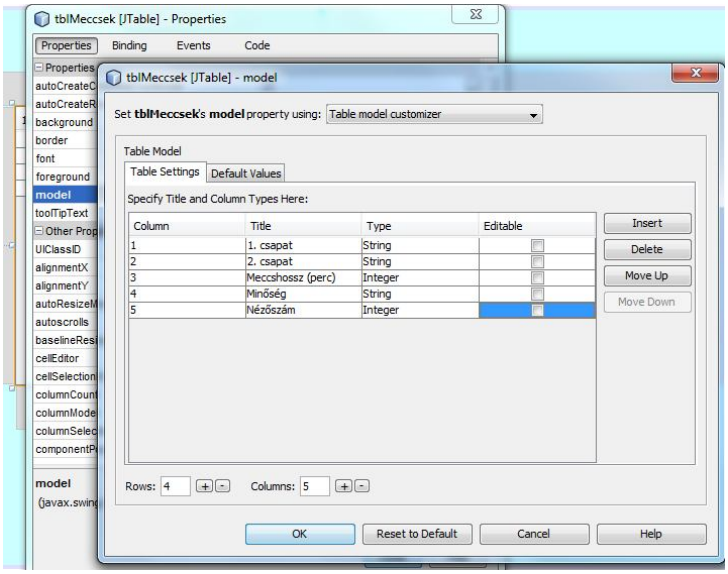
(De ezt csak az után célszerű megírni, hogy felrakta a paneleket a frame felületére.)

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

    private static final String CHAR_SET = "UTF-8";
    private boolean beolvasva;

    public SzabalyzatPanel() {
        initComponents();
        txtSzabalyzat.setEditable(false);
    }

    private void formAncestorAdded(javax.swing.event.AncestorEvent evt) {
        if (!beolvasva) {
            try (InputStream ins
                    = getClass().getResourceAsStream("/adatok/szabalyzat.txt");
                    Scanner fajlScanner = new Scanner(ins, CHAR_SET)) {
                String sor;
                while (fajlScanner.hasNextLine()) {
                    sor = fajlScanner.nextLine() + "\n";
                    txtSzabalyzat.append(sor);
                }
                // a szöveg elejére pozicionál
                txtSzabalyzat.setCaretPosition(0);
                beolvasva = true;

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

`MegnezettMeccsekPanel`:

A felület tervezési mérete szintén ugyanakkora, mint a másik két panelé. Erre kerül fel a label, a táblázat és a gomb (a táblázat neve: `tblMeccsek`, a gombé: `btnFajlba`). A táblázatot a palettáról is ki lehet választani (Swing Controls – Table), de persze, úgy is megoldható, ahogy a bemelegítő feladatban látta.

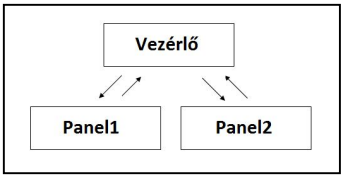
Most azt a változatot beszéljük meg, amikor a palettáról választjuk a táblázatot. A felületre felrakott táblázat modell tulajdonságát be tudjuk állítani (Properties/model – és itt a gombot kell megnyomni). Ekkor megadható, hogy hány oszlop legyen, mik legyenek az oszlopnevek, milyen típusú adatok legyenek bennük, stb. Természetesen sok egyéb tulajdonságot is be lehet állítani, akár a tulajdonságok ablakból, akár kódból, ha érdekli, járjon utána.



Ezeket a paneleket kell felraknunk a frame felületén lévő tabulált felületre. Ahogy korábban már szó volt róla, célszerű akkor, amikor még nem írtunk saját kódot. (Kivéve persze a már működő foci_EB_2 projekt kódját, de az nyilván helyesen fordul.) Ahogy felrakjuk a paneleket, megjelennek a kartotékfülek, amelyek feliratát persze át is lehet írni. Csak hogy arra is lásson példát, egyet kódból nevezünk majd át.

A táblázatba azok a meccsek kerülnek, amelyeket a `MeccsPanel` példány „Meccs” feliratú gombja hatására keletkeztek. A kérdés az, hogyan teremthetünk kapcsolatot a két panel között.

A földhözragadt megoldás az, hogy az egyik panel közvetlenül üzen a másiknak (azaz közvetlenül meghívja annak megfelelő metódusait), és esetleg viszont. Ez persze akár meg is oldhatja az aktuális feladatot, de megint az a baj vele, hogy nem hagy elég teret az esetleges későbbi bővítés számára. Az újrahasznosíthatóság szempontjából az a jó, ha minél függetlenebbek egymástól az egyes modulok, azaz esetünkben a két panel. Ahogy korábban már volt szó róla, az lenne a jó megoldás, ha továbbra is az MVC szemlélet szerint próbálnánk gondolkozni, vagyis mindkét panelnek csak a megjelenítés lenne a dolga, és a vezérlő osztály mondja meg, hogy mit is jelenítsenek meg. Ily módon könnyen lehetne változtatgatni a megjelenítő felületeket.



Vagyis ahogy az ábrán is látható, a két panel független egymástól, és mindketten csak a vezérlővel vannak kapcsolatban.

Ez első hallásra talán bonyolultnak tűnik, de tapasztalhatja majd, hogy jóval kényelmesebbé is teszi a programozást, mert így minden lényeges funkció „egy kézben” összpontosul.

Még egy fontos kérdés: honnan ismeri majd egymást a vezérlő és az adott panel? A vezérlőnek mindkét panelt ismernie kell, vagyis csak ott lehet „bemutatni” őket egymásnak, ahol együtt szerepel mindkettő. Ez a hely a frame. Előbb azonban fel kell készíteni az osztályokat arra, hogy egyáltalán „megismerkedhessenek” egymással. Ez a következőt jelenti:

A vezérlő osztálynak most két panelt kell irányítania, vagyis a `meccsPanel` példány mellett deklarálni kell még egy `MegnezettMeccsekPanel` típusú példányt is, és biztosítani, hogy ez is kaphasson értéket. Ezt megoldhatjuk úgy, hogy meghagyjuk a korábbi konstruktort és az újonnan deklarált panel példány egy setterrel kap értéket, de talán kicsit kényelmesebb, ha ezt is hozzávesszük a konstruktor paramétereihez. Ez utóbbi megoldás mellett az szól, hogy ekkor kevésbé lehet elfelejteni, hogy átadjuk a paneleket a vezérlőnek.

A paneleknek pedig azt kell tudniuk, hogy ki a vezér. Ezt ugyancsak lefordítva azt jelenti, hogy a paneleknek pedig rendelkezniük kell egy-egy `Vezerlo` típusú adattaggal. (Persze, csak akkor, ha a panel is üzenni akar a vezérlőnek. Ha a feladat csak annyi lenne, hogy a táblázatba írassuk ki a meccseket, akkor erre nem is lenne szükség, de mivel fájlba is akarjuk írni őket, ezért erre majd ismét a vezérlést kell megkérni.)

A paneleken deklarált `vezerlo` példány csak setterrel kaphat értéket, hiszen ahhoz, hogy a panelt rá tudjuk húzni a frame-re, paraméter nélküli konstruktorral kell rendelkeznie. (Vagy ha nem, akkor csak kódból tudjuk egymáshoz rendelni őket, ami kicsit kényelmetlenebb.)

Hogyan kerülnek be a meccsek a táblázatba? Erre két megoldást is mutatunk. Az első bonyolultabb, de nem azért kezdjük ezzel, hogy elmenjen a kedve, hanem azért, mert ez az általánosabb megoldás. A mostani feladatkiírásban nincs ilyen lehetőség, de elvileg előfordulhatna, hogy közvetítenek egy meccset, közben megnézzük, hogy szerepel-e a táblázatban, és hányan látták, majd visszatérünk a másik panelre, és kis idő múlva ismét ránézünk a táblázatra. Könnyen lehet, hogy közben megváltozott az adott meccs nézőszáma. Vagyis sok esetben lehet szükség arra, hogy frissítsük a táblázatot (a mostani feladat ezt nem igényli, a második megoldásban ezt az egyszerűbb változatot is megmutatjuk).

A táblázat akkor frissül, ha rákattintunk a „Megnézett meccsek” feliratú fülre. Ez az esemény a frame felületén keletkezik, de csak annyi a dolga, hogy közli a vezérlővel, hogy bekövetkezett. A többi a vezérlő dolga. Ez egyúttal azt is jelenti, hogy a `vezerlo` példányra több metódusnak is szüksége van, vagyis már nem elég lokálisan deklarálni. A megbeszéltek alapján a `FoFrame` osztály generált kódok nélküli részlete:

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

    private final int SZELESSEG = 816;
    private final int MAGASSAG = 588;
    private final String CIM = "Foci EB";

    private Vezerlo vezerlo;

    public FoFrame() {
        initComponents();
        beallitas();
    }

    private void beallitas() {
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(CIM);
        this.setLocationRelativeTo(null);
        jTabbedPane1.setTitleAt(2, "Megnézett meccsek");
    }



    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() {
        setVisible(true);
        vezerlo = new Vezerlo(meccsPanel1, megnezettMeccsekPanel1);
        meccsPanel1.setVezerlo(vezerlo);
        megnezettMeccsekPanel1.setVezerlo(vezerlo);
        vezerlo.indit();
    }

    private void megnezettMeccsekPanel1AncestorAdded(javax.swing.event.AncestorEvent evt) {
        vezerlo.meccseketTablazatba();
    }
```

A `Vezerlo` osztály régi változatába egyetlen apró dolgot kell beszúrnunk, mégpedig azt, hogy a generált meccsek kerüljenek be egy listába:

```
private List<Meccs> nezettMeccsek = new ArrayList<>();

public void meccs() {
        ...
    meccsPanel.kiirMeccs(meccs);
    nezettMeccsek.add(meccs);
        ...
}
```

A frame-ből meghívott metódus:

```
public void meccseketTablazatba() {
    megnezettMeccsekPanel.meccseketTablazatbaIr(nezettMeccsek);
}
```

A táblázatba a meccset játszó csapatok neve kerül, a meccs időtartama és minősége – ezeket eddig is tudtuk, de még azt is ki szeretnénk íratni, hogy hány házaspár nézte. Ehhez kicsit módosítanunk kell az alaposztályokat.

A `Meccs` osztályban azt is figyeljük, hogy hányan nézik az illető meccset, illetve írjunk egy metódust a minőség egyszerű kijelzésére:

```
private int nezoSzam;

public void nezik(){
    this.nezoSzam++;
}

public String getMinoseg() {
    return isJo() ? "jó" : "rossz";
}

public int getNezoSzam() {
    return nezoSzam;
}
```

A `Hazaspar` osztály `meccsNezes()` metódusa így módosul:

```
public void meccsNezes(Meccs meccs){
    ferj.meccsNezes(meccs);
    feleseg.meccsNezes(meccs);
    meccs.nezik();
}
```

A `MegnezettMeccsekPanel` osztály (generált kódok nélkül):

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

    private Vezerlo vezerlo;

    private DefaultTableModel tablaModel;

    public MegnezettMeccsekPanel() {
        initComponents();

        tblMeccsek.setShowGrid(true);

        // Mivel a palettáról leszedett tábla modelljét már létrehoztuk akkor,
        // amikor beállítottuk a tábla fejlécét, ezért most ezt a modellt
        // akarjuk használni.
        // A getModel() metódus TableModel típust ad vissza, ezért
        // típuskényszerítés kell (ha nem akarunk saját modell osztályt írni).
        tablaModel = (DefaultTableModel) tblMeccsek.getModel();
    }

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

    public void meccseketTablazatbaIr(List<Meccs> nezettMeccsek) {

        // Kitöröljük a tábla korábbi sorait
        int n = this.tablaModel.getRowCount();
        for (int i = n-1; i >= 0; i--) {
            this.tablaModel.removeRow(i);
        }

        // Létrehozzuk az új sorokat.
        for (Meccs meccs : nezettMeccsek) {

            Object[] tablaSor = {meccs.getCsapat1(), meccs.getCsapat2(),
                                 meccs.meccsHossz(), meccs.getMinoseg(),
                                 meccs.getNezoSzam()};

            tablaModel.addRow(tablaSor);
        }
    }
```

Megjegyzés: Figyelje meg a törlést. Ez egy meglehetősen veszélyes művelet, hiszen ha fordított sorrendben csinálnánk, akkor módosítanánk a ciklus tartományát, ami óhatatlanul elszálláshoz vezet. Ezért csak föntről lefelé szabad törölni.

Beszéljük meg a táblázatba írás másik lehetséges módját. A jelen feladat úgy szólt, hogy a középső panel „Meccs” feliratú gombjának hatására az összes házaspár eldönti, hogy nézi-e a meccset vagy sem, utána semmi változás nem történik. Vagyis a meccs generálása (és esetleg

a megnézése) után azonnal ki lehet íratni az aktuális meccset a táblázatba. Ez esetben a frame felületén nem kell semmilyen eseményt kezelni, vagyis a `Vezerlo` típusú példányt elég továbbra is lokálisan deklarálni a `start()` metódusban. Ha csak a táblázatba írás lenne a feladat, akkor a `Vezerlo` osztályban most nem is volna szükség a megnézett meccsek listájára, hiszen a generált meccset azonnal kiíratjuk, és nincs vele több dolgunk. Hogy mégis szükség van a listára, annak az az oka, hogy ennek a listának az elemeit írjuk majd ki az eredményfájlba.

A táblázatba írás másik módja tehát a következő: A `Vezerlo` osztály `meccs()` metódusának végére ez kerül:

```
megnezettMeccsekPanel.tablazatbaIr(meccs);
```

A `MegnezettMeccsekPanel` osztályban:

```
public void tablazatbaIr(Meccs meccs) {
    Object[] tablaSor = {meccs.getCsapat1(), meccs.getCsapat2(),
        meccs.meccsHossz(), meccs.getMinoseg(),
        meccs.getNezoSzam()};

    tablaModel.addRow(tablaSor);
}
```

Ha ezt a megoldást választjuk, akkor – mivel nem kell a frissítés miatt minden alkalommal kitörölni a korábbi sorokat – nincs szükség a törlés ciklusra. Ezért a design módban úgy kell felraknunk a táblázatot, hogy ne legyen egyetlen üres sora sem (különben csak az üres sorok után kezdi el feltölteni).

A gombnyomás hatására megnyílik majd egy fájlválasztó ablak, abban tudjuk megadni azt a fájlt, amelybe menteni akarjuk az adatokat. Ha a fájl kiválasztása után megnyomjuk a „save” gombot, akkor megvizsgálja, hogy létezik-e már ilyen fájl. Ha igen, akkor „illik” figyelmeztető üzenetet adni. Ha felülírható a fájl, illetve ha még egyáltalán nincs ilyen, akkor megkérjük a vezérlőt, hogy írja ki az adatokat az adott fájlba:

```
    private void btnFajlbaActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fajlValaszto = new JFileChooser(new File("."));
        if (fajlValaszto.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
            try {
                File fajl = fajlValaszto.getSelectedFile();
                if (fajl.exists()) {

                    if (JOptionPane.showConfirmDialog(null,
                            "a fájl már létezik, felülírjam?", "hibaüzenet",
                            JOptionPane.YES_NO_OPTION)
                                                    == JOptionPane.YES_OPTION) {
                        vezerlo.fajlbaIras(fajl);
                    }
                } else {
                    vezerlo.fajlbaIras(fajl);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
```

A fájlba írás a `Vezerlo` osztály dolga:

```
    public void fajlbaIras(File fajl) {
        try(PrintWriter kiiro = new PrintWriter(fajl)) {
            for (Meccs meccs : nezettMeccsek) {
                kiiro.println(String.format("%s;%s;%d;%s;%d",
                                        meccs.getCsapat1(), meccs.getCsapat2(),
                                        meccs.meccsHossz(), meccs.getMinoseg(),
                                        meccs.getNezoSzam()));
            }
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```