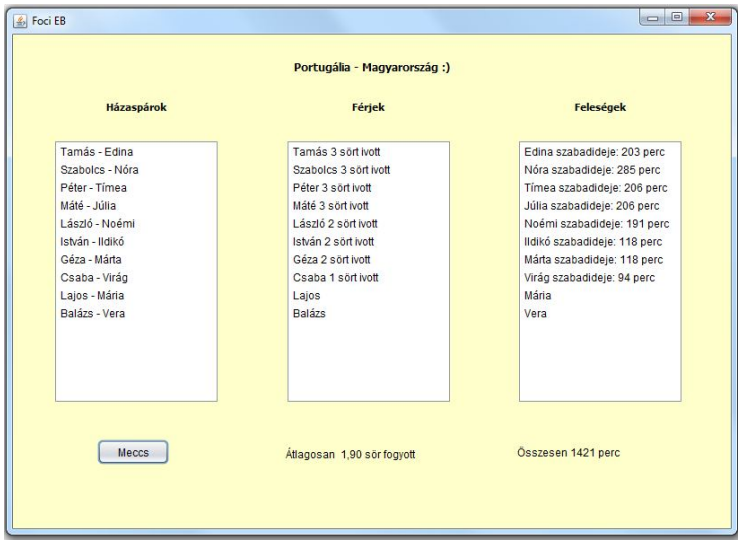
Foci EB 2.

Folytassuk a korábbi feladatot, és most grafikus felületen szimuláljuk a foci EB-t.

Mint korábban szó volt róla, a szimulációban házaspárok vesznek részt. A programban résztvevő minden egyes **ember**nek van *neve*, és mindegyikükre jellemző a *meccsnézés()*, de teljesen eltérő módon, az viszont közös, hogy ennek során a megnézett meccs (vagyis a metódus paramétere) bekerül a *megnézett meccsek* listájába.

Olvassa be a házaspárok és a csapatok adatait, de most úgy oldja meg a feladatot, hogy bármikor könnyen és gyorsan át lehessen váltani az adatfájlból való olvasásról adatbázisból való olvasásra, és viszont. A program indulásakor írassuk ki a házaspárok névsorát, illetve külön-külön a férjeket is és a feleségeket is. Induláskor még ne legyen látható a sörök és szabadidők értéke, ezek csak akkor jelennek meg, ha az illető személyek már láttak legalább egy meccset. Az átlag, összeg és persze, az aktuális meccs sem látszik induláskor, ezek csak az első meccsnézés után válnak láthatókká.

A Meccs feliratú gomb hatása:



A panel tetején megjelenik az aktuális meccs – ez úgy áll elő, hogy véletlenszerűen kiválasztjuk a meccshez tartozó két csapatot (figyeljen rá, hogy egy csapat ne játsszon saját magával, de persze, több meccs is lehet ugyanazon két csapat között). Állítsuk be a meccs ráadás-idejét – ez egy 0 és adott határ közötti véletlen érték, majd azt is, hogy jó-e a meccs – ennek esélye valahány százalék.

Ezek után a házaspárok véletlenszerűen „eldöntik”, hogy nézik-e a meccset – valahány százalék az esélye annak, hogy igen. Ennek hatására a férjekhez és feleségekhez kiírt adatok is változnak, illetve az átlag és összeg értékek is. A férjek, feleségek a két külön listában is „egymás mellett” legyenek, lehetőleg a házaspár férj tagjának sörszáma szerinti csökkenő sorrendben. A férjek névsora alatt jelenjen meg az összes férjre vonatkoztatott átlagosan fogyasztott sörszám, a feleségek névsora alatt pedig a feleségek összes szabadideje.

A felület belső mérete: 800*550 pixel.

Megoldásrészletek:

Célszerű a felület kialakításával kezdeni a feladat megoldását, hiszen minél kevesebb saját kód van a projektben, annál nagyobb eséllyel tudjuk hibátlanul lefordítani – és ez kell ahhoz, hogy az elkészített panelt rá tudjuk húzni a frame felületére. Készítse el tehát a kívánt felületű panel osztályt. A megoldásban használt elnevezések:

`lblMeccs` – erre a labelre írjuk ki az aktuális meccset;

`lstHazasparok`, `lstFerjek`, `lstFelesegek` – értelemszerűen a megfelelő listafelületek;

`btnMeccs` – a „Meccs” feliratú gomb;

`lblSorSzamAtlag`, `lblSzabadidoOsszeg` – az eredmények kiírásához szükséges label-ek.

Hogy ne kelljen visszalapozni a foci_EB_1 megoldásához, itt is közöljük az alaposztályok setterek, getterek nélküli kódját:

```
public class Csapat {

    private String nev;

    public Csapat(String nev) {
        this.nev = nev;
    }
```

```
public class Meccs {

    private Csapat hazai;
    private Csapat vendeg;

    private boolean jo;

    private static int jatekIdo;
    private int raadas;

    public Meccs(Csapat hazai, Csapat vendeg) {
        this.hazai = hazai;
        this.vendeg = vendeg;
    }

    public int meccsHossz(){
        return jatekIdo + raadas;
    }

    @Override
    public String toString() {
        return hazai.getNev() + " : " + vendeg.getNev();
    }
```

```java
public class Ember {
    private String nev;
    private List<Meccs> megnezettMeccsek = new ArrayList<>();

    public Ember(String nev) {
        this.nev = nev;
    }

    public void meccsNezes(Meccs meccs){
        if(!megnezettMeccsek.contains(meccs)){
            megnezettMeccsek.add(meccs);
        }
    }

    @Override
    public String toString() {
        return nev;
    }

    public List<Meccs> getMegnezettMeccsek() {
        return new ArrayList<>(megnezettMeccsek);
    }
```

```java
public class Feleseg extends Ember{

    private int szabadIdo;

    public Feleseg(String nev) {
        super(nev);
    }

    @Override
    public void meccsNezes(Meccs meccs) {
        super.meccsNezes(meccs);
        szabadIdo += meccs.meccsHossz();
    }

    @Override
    public String toString() {
        String temp = "";
        if (!super.getNezettMeccsek().isEmpty()) {
            temp = " szabadideje: " + szabadIdo + " perc";
        }
        return super.toString() + temp;
    }
```

```java
public class Ferj extends Ember{

    private int megivottSorokSzama;

    private static int joMeccsSorSzam;
    private static int rosszMeccsSorSzam;

    public Ferj(String nev) {
        super(nev);
    }

    @Override
    public void meccsNezes(Meccs meccs) {
        super.meccsNezes(meccs);
        if(meccs.isJo()) megivottSorokSzama += joMeccsSorSzam;
        else megivottSorokSzama += rosszMeccsSorSzam;
    }

    @Override
    public String toString() {
        String temp = "";
        if(!super.getNezettMeccsek().isEmpty()){
            temp = " " + megivottSorokSzama + " sört ivott";
        }
        return super.toString() + temp;
    }
```

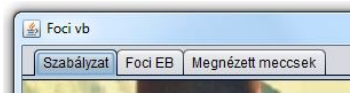
```java
public class Hazaspar implements Comparable<Hazaspar>{

    private Ferj ferj;
    private Feleseg feleseg;

    public Hazaspar(Ferj ferj, Feleseg feleseg) {
        this.ferj = ferj;
        this.feleseg = feleseg;
    }

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

    @Override
    public String toString() {
        return ferj.getNev() + " - " + feleseg.getNev();
    }

    @Override
    public int compareTo(Hazaspar t) {
        return t.ferj.getSorokSzama() - this.ferj.getSorokSzama();
    }
```

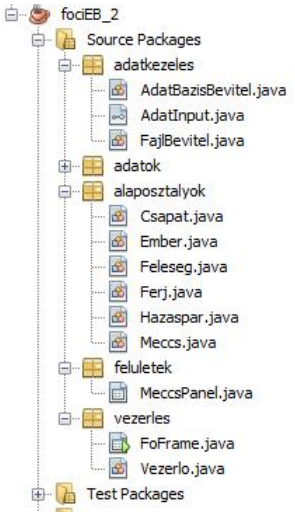
Az adatbevitelt megbeszéltük a foci_EB_1 feladat megoldásakor, erre most nem térünk ki, csak hivatkozunk a beolvasó metódusra.

A következő kérdés a programlogika szerkezete és helye.

Elvileg minden megoldható a panelen. Vagyis megírhatjuk itt az adatbeolvasást (ekkor a házaspárokat célszerű azonnal modellbe olvasni), a meccsek és meccsnézések generálását, összesítést, kiíratást, stb. Vagyis elvileg annyi is elég, ha az alaposztályok és a frame osztály mellett csak a panel osztály szerepel a projektben. Ez azonban nem a legszebb megoldás.

Az igazán szép megoldás az, ha az MVC szemléletnek megfelelően külön vezérlő osztályt használunk. Ez jóval könnyebbé teszi az esetleges későbbi módosítást, pl. azt, ha a továbbiakban még újabb panelekkel bővítjük a projektet.





Ekkor a vezérlő osztályban kell meghívni a beolvasást, és a paneleket csak megkérni rá, hogy jelenítsék meg az adatokat. Ez esetben azonban listába kellene raknunk a beolvasott adatokat (ahogy a korábbi megoldás is csinálja), mert ez általánosabb, mint a modell, és ezeket az adatokat esetleg más célra vagy más felületen is fel lehet használni.

Ha a meccseket is a vezérlő osztályban generáljuk, akkor gyakorlatilag néhány pillanat alatt el is készülhet a harmadik fül feladata, hiszen az csak annyi, hogy jelenítse meg a vezérlőben lévő meccsek listáját. Ez lesz tehát a megoldás során használt projekt-szerkezet:

A feladat szerint a program indulásakor már azonnal látjuk a beolvasott adatokat, vagyis a beolvasást vagy a panel formAncestorAdded eseményéhez kell rendelnünk, vagy a frame `main()` metódusából hívhatjuk. Ha MVC szemléletben szeretnénk megoldani a feladatot, akkor a panel dolga a megjelenítés, vagyis az a logikus, ha a `main()` metódus indítja az olvasást. Ez lényegében ugyanaz, mint amit konzolos esetben tettünk, ott is a `main()` metódusból hívtuk meg a vezérlő példány indító metódusát. Itt is ez a teendőnk, egyetlen plusz feladat van: láthatóvá is kell tennünk a felületet.

A frame általunk írt (vagyis nem generált) kódja:

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

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

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

```
    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 beallitas() {
        this.setSize(SZELESSEG, MAGASSAG);
        this.setTitle(CIM);
        this.setLocationRelativeTo(null);     }

    private void start() {
        setVisible(true);
        Vezerlo vezerlo = new Vezerlo(meccsPanel1);
        meccsPanel1.setVezerlo(vezerlo);
        vezerlo.indit();
    }
```

Mint látható, ebben a `start()` metódusban hozzuk létre a vezérlő példányt és ismertetjük össze egymással a vezérlőt és a panelt. Ha több panelt is használnánk, akkor azokat is itt kellene össze-ismertetni a vezérlővel.

A vezérlő ide vonatkozó metódusai – a rövidség kedvéért most a fájlból való olvasás kódját használjuk fel, de nyilván ugyanígy lehetne adatbázisból is olvasni.:

```
public class Vezerlo {

    private final int JATEK_IDO = 90;
    private final int JO_MECCS_SOR_DARABSZAM = 2;
    private final int ROSSZ_MECCS_SOR_DARABSZAM = 1;
    private final int MAX_RAADAS = 30;  // legföljebb ennyi a ráadás
    private final double JO_MECCS_SZAZALEK = 0.6;// ilyen százalékban jó a meccs
    private final double NEZES_SZAZALEK = 0.4;  // ekkora az esélye, hogy nézik

    private MeccsPanel meccsPanel;
    private String csapatUtvonal = "/adatok/csapatok.txt";
    private String hazasparUtvonal = "/adatok/parok.txt";
    private List<Hazaspar> hazasparok;
    private List<Csapat> csapatok;

    public Vezerlo(MeccsPanel meccsPanel) {
        this.meccsPanel = meccsPanel;
    }
```

```
public void indit() {
    statikusAdatok();
    try {
        adatBevitel();
    } catch (Exception ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE,null,ex);
    }
}

private void statikusAdatok() {
    Ferj.setJoMeccsSorSzam(JO_MECCS_SOR_DARABSZAM);
    Ferj.setRosszMeccsSorSzam(ROSSZ_MECCS_SOR_DARABSZAM);
    Meccs.setJatekIdo(JATEK_IDO);
}

private void adatBevitel() throws Exception {
    AdatInput adatInput = new FajlBevitel(csapatUtvonal, hazasparUtvonal);

    hazasparok = adatInput.hazasparBevitelLista();
    csapatok = adatInput.csapatBevitelLista();
    meccsPanel.adatMegjelenites(hazasparok);
}
```

Nyilván az adatok megjelenítése is feladat, ezt végzi a meccspanel `adatMegjelenites()` metódusa. Ahhoz, hogy ez működhessen, deklarálni kell a modelleket. Ezeket célszerű már a konstruktorban hozzárendelni a listafelületekhez.

Mivel a megoldás során a panelnek is ismernie kell a `Vezerlo` példányát, ezért azt is deklaráljuk, és megírjuk a hozzá tartozó settert is.

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

    private DefaultListModel<Hazaspar> hazasparModell = new DefaultListModel<>();
    private DefaultListModel<Ferj> ferjModell = new DefaultListModel<>();
    private DefaultListModel<Feleseg> felesegModell = new DefaultListModel<>();
    private Vezerlo vezerlo;

    public MeccsPanel() {
        initComponents();
        lstHazasparok.setModel(hazasparModell);
        lstFerjek.setModel(ferjModell);
        lstFelesegek.setModel(felesegModell);
    }
```

```
public void adatMegjelenites(List<Hazaspar> hazasparok) {
    hazasparModell.clear();
    ferjModell.clear();
    felesegModell.clear();
    for (Hazaspar hazaspar : hazasparok) {
        hazasparModell.addElement(hazaspar);
        ferjModell.addElement(hazaspar.getFerj());
        felesegModell.addElement(hazaspar.getFeleseg());
    }
}

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

Ha csak a beolvasott adatok megjelenítése volna a feladat, akkor nem lenne életbevágó takarítani a modelleket a feltöltés előtt, de ezt a metódust máskor is hívjuk majd (hiszen minden meccsnézés után frissíteni kell a felületet), ezért kell itt meghívnunk a modellek `clear()` metódusát.

A panel meccs gombjának hatása:

```
private void btnMeccsActionPerformed(java.awt.event.ActionEvent evt) {
    vezerlo.meccs();
}
```

A vezérlőben:

```
public void meccs() {
    // kiválasztunk két véletlen csapatot, arra figyelve, hogy ne legyenek
    // azonosak.
    int veletlenCsapatIndex1 = (int) (Math.random()*csapatok.size());
    int veletlenCsapatIndex2;
    do {
        veletlenCsapatIndex2 = (int) (Math.random() * csapatok.size());
    } while (veletlenCsapatIndex1 == veletlenCsapatIndex2);

    // létrehozunk egy meccset
    Meccs meccs = new Meccs(csapatok.get(veletlenCsapatIndex1),
            csapatok.get(veletlenCsapatIndex2));

    // véletlen értéket generálunk a hosszabbítás számára
    meccs.setRaadas((int) (Math.random() * Global.MAX_RAADAS));

    // ekkora eséllyel jó a meccs
    if (Math.random() < Global.JO_MECCS_SZAZALEK) {
        meccs.setJo(true);
    }
    meccsPanel.kiirMeccs(meccs);
```

```
        // ekkora eséllyel nézi egy-egy pár a meccset.
        for (Hazaspar hazaspar : hazasparok) {
            if (Math.random() < Global.NEZES_SZAZALEK) {
                hazaspar.meccsNezes(meccs);
            }
        }
        Collections.sort(hazasparok);
        meccsPanel.adatMegjelenites(hazasparok);
        double atlag = atlag();
        int osszeg = osszeg();
        meccsPanel.kiir(atlag, osszeg);
    }

    private double atlag() {
        double ossz = 0;
        for (Hazaspar hazaspar : hazasparok) {
            ossz += hazaspar.getFerj().getSorokSzama();
        }
        if (hazasparok.size()> 0) {
            return ossz/hazasparok.size();
        }
        return -1;
    }

    private int osszeg() {
        int ossz = 0;
        for (Hazaspar hazaspar : hazasparok) {
            ossz += hazaspar.getFeleseg().getSzabadIdo();
        }
        return ossz;
    }
```

A `MeccsPanel` két hivatkozott metódusa:

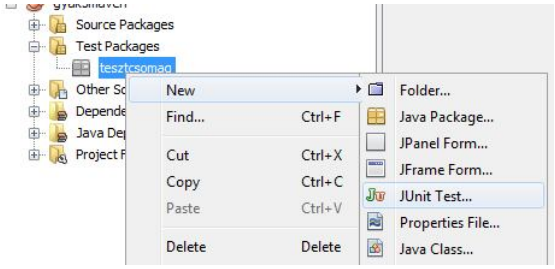
```
    public void kiir(double atlag, int osszeg) {
        if(atlag > -1){
            lblSorszam.setText(String.format("Átlagosan %5.2f sör fogyott",
                                                    atlag));
        }
        else lblSorszam.setText("Nincs adat");
        lblSzabadido.setText("Összesen " + osszeg + " perc");
    }

    public void kiirMeccs(Meccs meccs) {
        lblMeccs.setText(meccs.toString());
    }
```

Megjegyzés: Mivel rendezés után ugyanazt az adatmegjelenítő metódust hívtuk meg, mint a beolvasás során, ezért ha megváltozik a sorrend, akkor mindhárom listafelületen változik. Ha pl. a házaspárokat tartalmazó listafelületet nem akarjuk változtatni, akkor értelemszerűen nem ugyanezt a kiíratást kell meghívnunk.

Mivel itt az egyparaméteres `sort()` metódust hívtuk meg, ez csak akkor működhet, ha a `Hazaspar` osztály `Comparable` típusú, azaz implementálja a `compareTo()` metódust. (ld. följebb az osztály kódját)

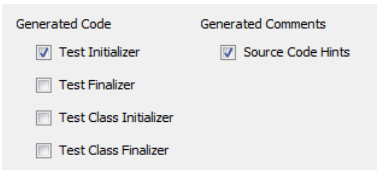
Még egy kis ráadás:

Bár remélhetőleg már saját belső igénye, hogy teszteket is írjon, de ha netaláltán megfeledkezett volna róla, akkor ismételjük át egy egyszerű, pici teszt segítségével.



Biztosan emlékszik, hogy így lehet generálni:

A Test Package mappában hozzunk létre egy csomagot, ebben pedig egy JUnit Test típusú fájlt.



Elég, ha ezek vannak bekapcsolva.

Ekkor létrejön egy JUnit teszt osztály. Ennek metódusai a teszt futtatásakor (sima Java projekt esetén fájlnév, jobb egérgomb, Run File, Maven esetén Test File) futnak le. A metódusok sorrendje esetleges, azokat futtatja, amelyek fölött ott van a `@Test` annotáció. A `@Before` annotációval jelölt metódusok pedig minden teszt-metódus futása előtt lefut, ezért ide írjuk pl. a beállításokat.

Egy nagyon egyszerű teszt osztály:

(Próbálja meg kiegészíteni még néhány saját teszt-ötlettel.)

```java
public class Teszt {

    public Teszt() {
    }

    private Ferj ferj;
    private Feleseg feleseg;
    private Hazaspar hazaspar;

    private Meccs meccs;

    private static final String FERJ_NEV = "Férjnév";
    private static final String FELESEG_NEV = "Feleségnév";

    private static final String HAZAI = "Hazai";
    private static final String VENDEG = "Vendég";

    private static final int JO_SORSZAM = 2;
    private static final int ROSSZ_SORSZAM = 1;
    private static final int JATEKIDO = 90;
    private static final int RAADAS = 10;

    @Before
    public void setUp() {
        ferj = new Ferj(FERJ_NEV);
        feleseg = new Feleseg(FELESEG_NEV);
        hazaspar = new Hazaspar(ferj, feleseg);

        meccs = new Meccs(new Csapat(HAZAI), new Csapat(VENDEG));

        Ferj.setJoMeccsSorDb(JO_SORSZAM);
        Ferj.setRosszMeccsSorDb(ROSSZ_SORSZAM);
        Meccs.setJatekIdo(JATEKIDO);
        meccs.setRaadas(RAADAS);
    }

    @Test
    public void fociTeszt() {
        // Csak néhány egyszerű példa

        // még nem láttak egyetlen meccset sem:
        assertTrue(hazaspar.getFerj().getMegnezettMeccsek().size() == 0);

        meccs.setJo(true);

        hazaspar.meccsNezes(meccs);

        // már egy meccset látott:
        assertTrue(hazaspar.getFerj().getMegnezettMeccsek().size() == 1);

        assertTrue(hazaspar.getFeleseg().getMegnezettMeccsek().size() == 1);

        assertTrue(hazaspar.getFerj().getSorokSzama() == JO_SORSZAM);

    }
}
```