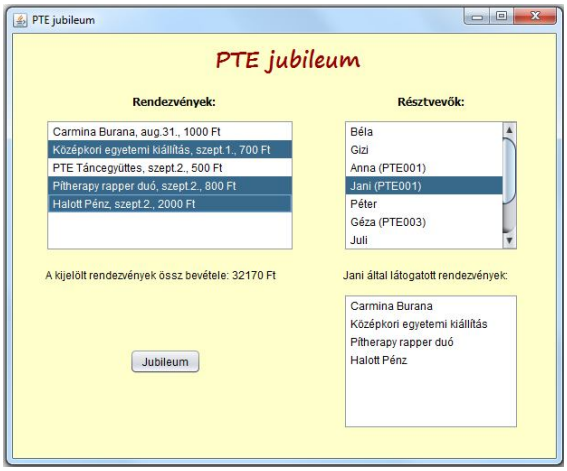
Jubileumi feladat grafikus felületen

Alakítsuk át a múltkori megoldást grafikus felületű alkalmazássá, mégpedig úgy, hogy lehetőleg minél többet használjunk a meglévő megoldásból.



Az ábrán látható külsejű alkalmazás a következőket tudja:

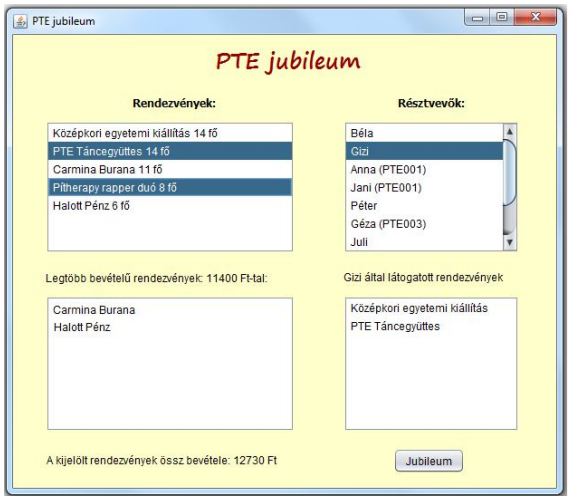
A program indulásakor már legyenek olvashatóak a rendezvények és a résztvevők listájának adatai, illetve adatbeolvasáskor minden résztvevő kapjon valamennyi pénzt.

A Jubileum feliratú gomb megnyomására fusson le az a szimuláció, amely már korábban is feladat volt, vagyis az, hogy minden rendezvény esetén mindegyik résztvevő „döntse el” véletlenül, hogy részt akar-e venni a rendezvényen vagy sem.

A rendezvények listája alatt lehessen látni a kijelölt rendezvények össz-bevételét.

A résztvevők listája alatt pedig azt, hogy a kiválasztott résztvevő mely rendezvényeken vett részt. Mindkét felirat csak akkor jelenjen meg, ha már választottunk a megfelelő listából.

Bővítsük a feladatot: a Jubileum gomb hatására jelenjen meg a legtöbb bevételt hozó rendezvények listája is, maguk a rendezvények pedig legyenek résztvevőszám szerint csökkenően kiírva (most úgy, hogy a rendezvény neve és a létszám).



Megoldás:

A grafikus felületet a NetBeans-ben generáltuk, a generált részleteket nem másolom ide. A felület így alakul ki:

- Egy `JFrame` típusú osztály – szerepe a program keretbe foglalása, vagyis ez adja meg a felület grafikus keretét, de ez indítja a program futását is, ugyanis itt generálódik a `main()` metódus. (De ha több frame is van egy programban, akkor a generált `main()` metódust csak abban a frame-ben kell meghagyni, amelyet vezérlőnek szánunk.)
- Erre kerül fel egy (vagy több) panel, de a könnyebb módosíthatóság elérése céljából ezeket is saját, `JPanel` típusú osztályként érdemes megoldani.

Mivel ezt a feladatot már megoldottuk konzolos környezetben, ezért ezt a programot célszerű úgy megírni, hogy minél többet tudjunk felhasználni a már megírt kódból.
Az alaposztályok gyakorlatilag szó szerint átmásolhatók (csak a `toString()`-et írtuk át az újonnan kért alakra).
A vezérlést nyilván kicsit jobban át kell írni, de a lényege annak is megmarad. A konzolos alkalmazás a megadott sorrendben meghívott metódusok végrehajtását jelenti. Egy grafikus felületű alkalmazást események irányítanak. Vagyis a szóban forgó metódusokat egy-egy esemény bekövetkezése indítja el. Maguk a metódusok szinte módosítás nélkül használhatóak.
Az adott feladat megoldása során egyetlen dolgot módosítottunk a korábban megírt metódusokban, mégpedig azt, hogy az objektumokat nem `List` típusú gyűjteményben tároljuk, hanem modellekben. Ezek is gyűjtemények, vagyis sok hasonlóság van köztük és a korábban tanult listák között, csak valamivel kevesebb metódus tartozik hozzájuk, viszont vannak bennük olyanok, amelyek azért felelnek, hogy minden változásról értesítsék a hozzájuk tartozó lista-felületet. Egy `JList` típusú felülethez többféle listamodell tartozhat (persze, egyszerre csak egy), egyelőre a `DefaultListModel`-t használtuk.

Ennyi bevezető után azt gondolom, elég idemásolni a létrehozott kódot (a generált részek és a korábban megírt alaposztályok nélkül).

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

    private int szelesseg = 616;
    private int magassag = 538;
    private String cim = "PTE jubileum";

    public JubileumFrame() {
        initComponents(); // fontos: mindig ez az első
        this.setSize(szelesseg, magassag);
        this.setTitle(cim);
        this.setLocationRelativeTo(null);
    }
```

A generált `main()` metódusból egyelőre csak ez a lényeg:

```
new JubileumFrame().setVisible(true);
```

vagyis az, hogy példányosítja, és láthatóvá teszi a frame-t.

A frame osztálynak jelenleg csak ennyi a szerepe.

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

    private final double KEDVEZMENY_SZAZALEK = 10;
    private final String RENDEZVENY_ELERES = "rendezvenyek.txt";
    private final String RESZTVEVOK_ELERES = "resztvevok.txt";
    private final int ALSO_PENZ = 1000;
    private final int FELSO_PENZ = 10000;
    private final double KEDV_SZAZALEK = 0.8;

    private final String MAX_LABEL_SZOVEG = "Legtöbb bevételű rendezvények: ";

    private DefaultListModel<Rendezveny> rendezvenyModel = new DefaultListModel<>();
    private DefaultListModel<Resztvevo> resztvevoModel = new DefaultListModel<>();
    private DefaultListModel<String> resztvevoRendezvenyeModel = new DefaultListModel<>();
    private DefaultListModel<String> maxRendezvenyekModel = new DefaultListModel<>();

    public JubileumPanel() {
        initComponents();
        lstRendezvenyek.setModel(rendezvenyModel);
        lstResztvevok.setModel(resztvevoModel);
        lstResztvevokRendezvenyei.setModel(resztvevoRendezvenyeModel);
        lstMaxBevetel.setModel(maxRendezvenyekModel);
        lblMax.setText(MAX_LABEL_SZOVEG);
    }

    private void formAncestorAdded(javax.swing.event.AncestorEvent evt) {
        adatbevitel();
    }

    private void btnJubileumActionPerformed(java.awt.event.ActionEvent evt) {
        jubileum();
        maxKereses();
        rendezve();
    }

    private void lstResztvevokMousePressed(java.awt.event.MouseEvent evt) {
        Resztvevo resztvevo = (Resztvevo) lstResztvevok.getSelectedValue();
        lblResztvevo.setText(resztvevo.getNev() + " által látogatott rendezvények ");
        resztvevoRendezvenyeModel.clear();
        for (Rendezveny rendezveny : resztvevo.getRendezvenyek()) {
            resztvevoRendezvenyeModel.addElement(rendezveny.getCim());
        }
    }

    private void lstRendezvenyekValueChanged(javax.swing.event.ListSelectionEvent evt) {
        int ossz = 0;
        List<Rendezveny> valasztottak = lstRendezvenyek.getSelectedValuesList();
        for (Rendezveny rendezveny : valasztottak) {
            ossz += rendezveny.getBevetel();
        }
        lblRendezveny.setText(
                String.format("A kijelölt rendezvények össz bevétele: %d Ft", ossz));
    }
```

```
private void adatbevitel() {

    PTEsResztvevo.setKedvezmenySzazalek(KEDVEZMENY_SZAZALEK);

    try {
        Scanner fajlScanner = new Scanner(new File(RENDEZVENY_ELERES));
        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);
            rendezvenyModel.addElement(rendezveny);
        }
        fajlScanner.close();

        fajlScanner = new Scanner(new File(RESZTVEVOK_ELERES));
        Resztvevo resztvevo = null;
        while (fajlScanner.hasNextLine()) {
            sor = fajlScanner.nextLine();
            adatok = sor.split(";");
            if(adatok.length == 1) {
                resztvevo = new Resztvevo(adatok[0]);
            }
            if(adatok.length == 2) {
                resztvevo = new PTEsResztvevo(adatok[0], adatok[1]);
            }
            resztvevo.penztKap((int) (Math.random()*
                                (FELSO_PENZ - ALSO_PENZ + 1) + ALSO_PENZ));
            resztvevoModel.addElement(resztvevo);
        }
        fajlScanner.close();

    } catch (FileNotFoundException ex) {
        Logger.getLogger(JubileumPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void jubileum() {
    for (int i = 0; i < rendezvenyModel.size(); i++) {
        for (int j = 0; j < resztvevoModel.size(); j++) {
            if(Math.random() < KEDV_SZAZALEK)
                rendezvenyModel.get(i).resztVesz(resztvevoModel.get(j));
        }
    }
}
```

```
    private void maxKereses() {
        int max = rendezvenyModel.get(0).getBevetel();
        for (int i = 0; i < rendezvenyModel.size(); i++) {
            if(rendezvenyModel.get(i).getBevetel() > max){
                max = rendezvenyModel.get(i).getBevetel();
            }
        }

        lblMax.setText(MAX_LABEL_SZOVEG + max + " Ft-tal:");
        maxRendezvenyekModel.clear();
        for (int i = 0; i < rendezvenyModel.size(); i++) {
            if(rendezvenyModel.get(i).getBevetel() == max){
                maxRendezvenyekModel.addElement(rendezvenyModel.get(i).getCim());
            }
        }
    }

    private void rendezve() {
        List<Rendezveny> rendezvenyek =
                Collections.list(rendezvenyModel.elements());
        Collections.sort(rendezvenyek, new LetszamSzerint());

        Rendezveny.setRendezett(true);
        rendezvenyModel.clear();
        for (Rendezveny rendezveny : rendezvenyek) {
            rendezvenyModel.addElement(rendezveny);
        }
    }
}
```

Megjegyzés: Sajnos a default listamodell nem lehet a `sort()` metódus paramétere (később majd szó lesz másfajta, rendezhető modellről is), ezért egyelőre csak „fapados” megoldást tudunk, mégpedig azt, hogy a modell elemeit átrakjuk egy `List` típusú listába, azt rendezzük, majd a lista elemeit visszarakjuk a modellbe. A modell elemeit a `Collections` osztály `list()` metódusával is átrakhatjuk, de ha ez nem megy, akkor még fapadosabb módon, vagyis egy közönséges `for` ciklus segítségével.

A megoldásból még egy lépés hiányzik: a rendezés után más külalakban íratjuk ki a rendezvényeket. A listafelületen az objektumok `toString()`-je jelenik meg. Ebből nyilván nem lehet kétféle, de az lehet, hogy egy feltételtől függően más-más szöveget adjon vissza.
Ennek megoldása a `Rendezveny` osztályban:

```
    private static boolean rendezett;

    @Override
    public String toString() {
        if(rendezett) return cim + " " + resztvevokSzama + " fő";
        return cim + ", " + idoPont + ", " + jegyAr + " Ft";
    }

    public static void setRendezett(boolean rendezett) {
        Rendezveny.rendezett = rendezett;
    }
```