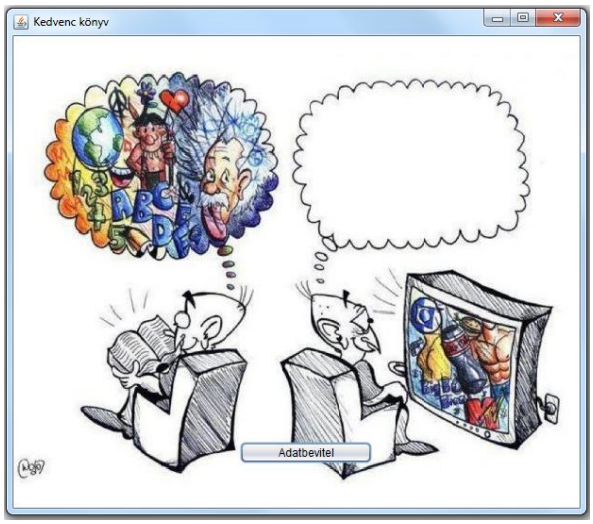
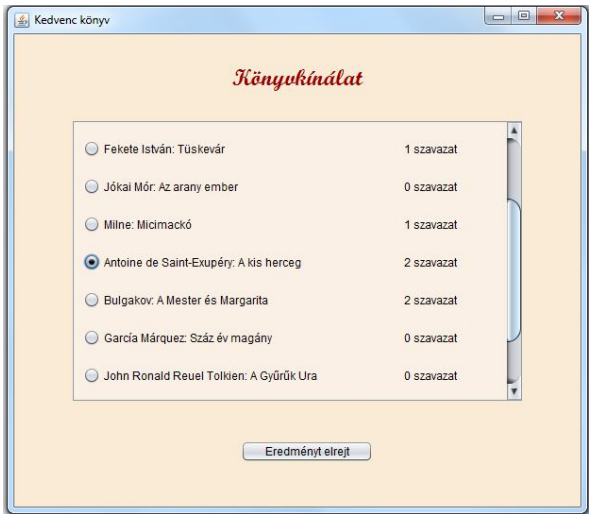
Kedvenc könyv

Válasszuk meg az év kedvenc könyvét! A 600 x 500-as belső méretű alkalmazás az alábbi képen látható módon indul.

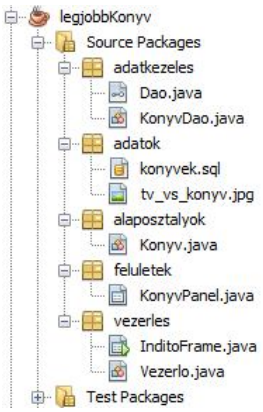


Az Adatbevitel gomb hatására adatbázisból beolvassuk a könyvek adatait (legelső alkalommal még egyáltalán nincsenek szavazatok, későbbi betöltéskor már a korábbi szavazatok értékét is beolvassuk). Sikeres beolvasás esetén az alábbiak közül a baloldali alakúra változik a felület. Egy-egy könyvre úgy tudunk szavazni, hogy megnyomjuk a hozzá tartozó rádiógombot. A gomb felirata közben „Eredményt mutat”-ra változik. Ha megnyomjuk, akkor láthatjuk a szavazás állását, a gomb felirata pedig „Eredményt elrejt”-re változik. Most megnyomva elrejtjük az eredményeket. A gombok megnyomása után a korábbi kiválasztás sem látszódik.



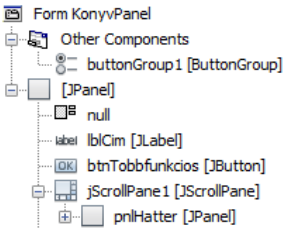
A szavazás eredménye azonnal kerüljön be az adatbázisba.

Megoldásrészletek:



Egy lehetséges projekt-szerkezet:

A `KonyvPanel` osztály „kézzel” kialakított szerkezete:



vagyis a panelre felkerül a főcím, a gomb, illetve középre egy `JSrollPane` típusú példány, ebbe pedig egy panel. Ezen kívül rakjunk fel egy `ButtonGroup` típusú példányt is. (Az elnevezések látszódnak az ábrán).

A panelt célszerű null layout módon beállítani (azért, hogy a felrakott komponensek látható vagy láthatatlan mivolta ne befolyásolja a gomb helyzetét).

A `Konyv` osztály (getterek nélkül):

```
public class Konyv {

    private String szerzo;
    private String cim;
    private String isbn;
    private int szavazatSzam;

    public Konyv(String szerzo, String cim, String isbn) {
        this.szerzo = szerzo;
        this.cim = cim;
        this.isbn = isbn;
    }

    public Konyv(String szerzo, String cim, String isbn, int szavazatSzam) {
        this.szerzo = szerzo;
        this.cim = cim;
        this.isbn = isbn;
        this.szavazatSzam = szavazatSzam;
    }

    public void szavazatotKap(){
        szavazatSzam++;
    }

    @Override
    public String toString() {
        return szerzo + ": " + cim ;
    }

    public String szavazatSzamSzoveg(){
        return szavazatSzam + " szavazat";
    }
```

Azért került két konstruktor az osztályba, mert az adatbázis adataiból a négyparaméteres konstruktor alapján hozunk létre példányt, de elképzelhető, hogy egy esetleges későbbi bővítés során még utólag is (vagyis a kezdeti adatbázis létrehozása után, mondjuk egy másik felületen) lehet újabb könyveket indítani a versenyben. Ekkor az újonnan létrehozott könyveket a háromparaméteres konstruktor példányosítaná.

Az adatkezelést most is külön osztályban oldjuk meg, mégpedig most is úgy, hogy egy interfészt implementálunk, csak most követjük az úgynevezett DAO tervezési mintát. (ld. pl itt: https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm)

Az interfész általában a CRUD (create, read, update, delete) műveleteket tartalmazza, lehetőleg minél általánosabb módon (vagyis generikus típusok használatával), az ezt implementáló osztály pedig a konkrét típusra valósítja meg.

Esetünkben:

```
public interface Dao<Type, PrimaryKey extends Serializable> {

    public void create(Type t)throws Exception;

    public Type read(PrimaryKey id)throws Exception;

    public void update(Type t)throws Exception;

    public void delete(Type t)throws Exception;

    public List<Type> listAll() throws Exception;
}
```

A feladat szerint ezek közül jelenleg csak az update műveletre van szükségünk, ezért csak ezt valósítjuk meg az implementáló osztályban, de továbbfejlesztésként próbáljon meg legalább még egy műveletet megvalósítani.

Az alapműveleteken kívül viszont még szükségünk van az adatbázis összes elemének listázására is, ezért ezt a metódust is megírjuk az osztályon belül.

```
public class KonyvDao implements Dao <Konyv, String>{

    private Connection kapcsolat;

    public KonyvDao(Connection kapcsolat) {
        this.kapcsolat = kapcsolat;
    }

    @Override
    public void create(Konyv konyv) throws Exception {
        throw new UnsupportedOperationException("Not supported yet.");
    }
```

```
    @Override
    public Konyv read(String isbn) throws Exception {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void update(Konyv konyv) throws Exception {
        if (kapcsolat != null) {

// 1. változat
//            String sqlUtasitas = "UPDATE KONYVEK set szavazatszam = "
//                                 + konyv.getSzavazatSzam()
//                                 + " WHERE isbn = '"
//                                 + konyv.getIsbn() + "'";
//            try(Statement utasitasObj = kapcsolat.createStatement()){
//                utasitasObj.executeUpdate(sqlUtasitas);
//            }

// 2. változat
//                StringBuilder sqlBuilder =
//                        new StringBuilder("UPDATE KONYVEK set szavazatszam = ");
//                sqlBuilder.append(konyv.getSzavazatSzam()).append("WHERE isbn = '");
//                sqlBuilder.append(konyv.getIsbn()).append("'");
//
//                try(Statement utasitasObj = kapcsolat.createStatement()){
//                    utasitasObj.execute(sqlBuilder.toString());
//                }

// 3. változat
            String sqlUtasitas = "UPDATE KONYVEK set szavazatszam = ? WHERE isbn = ?";

            try (PreparedStatement parameteresUtasitasObjektum
                    = kapcsolat.prepareStatement(sqlUtasitas)) {
                parameteresUtasitasObjektum.setInt(1, konyv.getSzavazatSzam());
                parameteresUtasitasObjektum.setString(2, konyv.getIsbn());

                parameteresUtasitasObjektum.executeUpdate();
            }
        }
    }
```

Mielőtt folytatnánk, néhány szó az `update()` metódusról:

A metódusban három változat is szerepel. Mindháromnak ugyanaz a hatása, de talán a 3. változat a legkevésbé sérülékeny. De milyen sérülékenységről is lehet szó? Bár ez elsősorban webes alkalmazások problémája lehet, de nem árt szót ejteni arról, hogy az ügyetlenül megadott SQL utasításokkal vissza is lehet élni. (Ld. pl.: http://sec2013.crysys.hu/blog/egy-halalos-injekcio/).

Erre leginkább akkor kerülhet sor, ha string-konkatenációval adjuk meg az utasítást (1. változat). Egy fokkal jobb, ha konkatenáció helyett felépítjük az utasítást, ahogy a 2. verzióban látható. (A felépítési módok különbségéről ld. pl.: https://stackoverflow.com/questions/2971315/string-stringbuffer-and-stringbuilder)

A harmadik változatban paraméteresen adjuk meg, hogy hova melyik érték kerül. (Ld. pl.: http://tutorials.jenkov.com/jdbc/preparedstatement.html)

Mindegy, hogy az `execute()` vagy az `executeUpdate()` metódust használjuk (az `execute()` a másik általánosítása).

Folytassuk az osztály kódját:

```
    @Override
    public void delete(Konyv konyv) throws Exception {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public List<Konyv> listAll() throws Exception {
        List<Konyv> konyvek = new ArrayList<>();
        String sqlUtasitas = "SELECT * from KONYVEK";
        if (kapcsolat != null) {
            try (Statement utasitasObjektum = kapcsolat.createStatement();
                    ResultSet eredmenyHalmaz =
                                utasitasObjektum.executeQuery(sqlUtasitas)) {

                String szerzo, cim, isbn;
                int szavazatszam;
                while (eredmenyHalmaz.next()) {
                    szerzo = eredmenyHalmaz.getString("szerzo");
                    cim = eredmenyHalmaz.getString("cim");
                    isbn = eredmenyHalmaz.getString("isbn");
                    szavazatszam = eredmenyHalmaz.getInt("szavazatszam");
                    konyvek.add(new Konyv(szerzo, cim, isbn, szavazatszam));
                }
            }
        }
        return konyvek;
    }
}
```

Adatbázis kezeléskor tehát ezeket a metódusokat használjuk majd. Most viszont folytassuk a projekt többi részével.

A frame a megszokott: itt ismertetjük össze a vezérlő és a panel példányt.

A panelen kirajzoltatjuk/eltüntetjük a háttérképet. A gombnyomás hatása attól függ, hogy mikor nyomjuk meg: először az adatbevitelt hívja meg, majd az eredmények láthatóságát módosítja.

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

    private String[] gombFelirat =
                        {"Adatbevitel", "Eredményt mutat", "Eredményt elrejt"};
    private int katt;
    private Image kep =
            new ImageIcon(this.getClass().
                            getResource("/adatok/tv_vs_konyv.jpg")).getImage();

    private Vezerlo vezerlo;

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

    private void beallitas(){
        btnTobbfunkcios.setText(gombFelirat[0]);
        lathatosag(false);
    }

    private void lathatosag(boolean lathato) {
        lblCim.setVisible(lathato);
        jScrollPane1.setVisible(lathato);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(kep, 0, 0, this.getWidth(), this.getHeight(), null);
    }

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

    private void btnTobbfunkciosActionPerformed(java.awt.event.ActionEvent evt) {
        katt++;
        if(katt == 1){
            lathatosag(true);
            kep = null;
            this.repaint();
            vezerlo.adatBazisMegnyitas();
        }
        if(katt % 2 != 0){
           btnTobbfunkcios.setText(gombFelirat[1]);
           szavazatSzamLathato(false);
        }else{
            btnTobbfunkcios.setText(gombFelirat[2]);
            szavazatSzamLathato(true);
        }
    }
```

Az adatbevitel a vezérlő feladata, csakhogy most nem külső, valaki (vagy akár saját magunk) által korábban létrehozott adatbázissal dolgozunk, hanem legelső futtatáskor egy SQL fájlból hozzuk létre azt, később viszont a már így létrehozott, és a többszöri futás során módosított adatbázisból olvassuk be az adatokat. FONTOS: Figyelje meg a mellékelt sql fájlt, a sorok végén most nincs pontosvessző!

```
public class Vezerlo {

    private final String SQL_ELERES = "/adatok/konyvek.sql";
    private final String CHAR_SET = "UTF-8";

    private KonyvDao dao;
    private KonyvPanel konyvPanel;
    private List<Konyv> beolvasottKonyvek;

    public Vezerlo(KonyvPanel konyvPanel) {
        this.konyvPanel = konyvPanel;
    }

    public void adatBazisMegnyitas() {
        try {
            dao = new KonyvDao(kapcsolodas());
            beolvasottKonyvek = dao.listAll();
            konyvPanel.konyveketKiIr(new ArrayList<>(beolvasottKonyvek));
        } catch (Exception ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private Connection kapcsolodas() throws ClassNotFoundException, SQLException {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        String url = "jdbc:derby:KonyvDB;create=true;";

        Connection kapcsolat = DriverManager.getConnection(url);

        String sqlVaneMarTabla =
                "select * from SYS.SYSTABLES where tablename = 'KONYVEK'";

        try (Statement utasitasObj = kapcsolat.createStatement();
                ResultSet rs = utasitasObj.executeQuery(sqlVaneMarTabla)) {

            if (!rs.next()) {
                adatBazisLetrehozas(utasitasObj);
            }
        }
        return kapcsolat;
    }
```

```
private void adatBazisLetrehozas(Statement utasitasObj) throws SQLException {
    try(InputStream ins = this.getClass().getResourceAsStream(SQL_ELERES);
            Scanner sc = new Scanner(ins, CHAR_SET)){
        String sor;
        while(sc.hasNextLine()){
            sor = sc.nextLine();
            if(!sor.isEmpty()){
                utasitasObj.execute(sor);
            }
        }
    } catch (IOException ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Látható, hogy beolvasáskor a `dao` példány `listAll()` metódusát használtuk, illetve megkérjük a `konyvPanel` példányt, hogy hozza létre, és jelenítse meg az egyes könyvekhez tartozó rádiógombokat.

A metódus ezt csinálja:

A beolvasott könyvek listáján végigiterálva minden egyes könyv esetén meghatározza a létrehozandó rádiógomb helyét és méretét, létrehozza a rádiógombot, és „rárakja” (`add()`) a korábban megadott `pnlHatter` nevű panelre. Ugyancsak rákerülnek a panelre (a megfelelő rádiógombok mellé) azok a labelek is, amelyekre a szavazatok száma kerül, ezek láthatóságát a gombnyomástól függően szabályozzuk majd. A használt méretek a metódus elején vannak beállítva. Ezek: a rádiógombok panel bal szélétől való vízszintes távolsága, az első rádiógomb függőleges távolsága a panel tetejétől, a rádiógombok egymástól való függőleges távolsága, a labelek vízszintes távolsága a panel bal szélétől, illetve a komponensek megadott méretei.

A rádiógombokhoz eseményfigyelőt is rendelünk, azt figyeli, hogy ki van-e választva az illető gomb. Ha igen, akkor a hozzá tartozó könyv szavazatot kap. Azt, hogy ez melyik indexű könyv, a rádiógomb pozíciójából állapítjuk meg.

Végül a háttérpanel méretét is be kell állítanunk. Ez nyilván függ attól, hogy hány rádiógomb kerül rá (azaz hány könyv van).

A `KonyvPanel` osztály most tárgyalt metódusa:

```
public void konyveketKiIr(List<Konyv> beolvasottKonyvek) {
    JRadioButton rdButton;
    JLabel label;
    int vizszintesTav = 10, fuggolegesTav = 20;
    int koztesTav = 40;
    int rdButtonSzelesseg = 300;
    int rdButtonMagassag = 20;

    int vizszintesLabelTav = 350;
    int labelSzelesseg = 100;
    int labelMagassag = 20;
    int scrollBarSzelesseg = 21;

    for (int i = 0; i < beolvasottKonyvek.size(); i++) {

        rdButton = new JRadioButton(beolvasottKonyvek.get(i).toString());
        rdButton.setLocation(vizszintesTav, fuggolegesTav + i * koztesTav);
        rdButton.setSize(rdButtonSzelesseg, rdButtonMagassag);
        rdButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                int index = (((JRadioButton) ae.getSource()).getY()
                                            - fuggolegesTav) / koztesTav;
                vezerlo.szavaz(index);
            }
        });
        this.pnlHatter.add(rdButton);
        buttonGroup1.add(rdButton);

        label = new JLabel(beolvasottKonyvek.get(i).szavazatSzamSzoveg());
        label.setLocation(vizszintesLabelTav, fuggolegesTav + i * koztesTav);
        label.setSize(labelSzelesseg, labelMagassag);
        this.pnlHatter.add(label);
    }
    Dimension preferredSize
            = new Dimension(jScrollPane1.getWidth() - scrollBarSzelesseg,
                    beolvasottKonyvek.size() * koztesTav + fuggolegesTav);

    pnlHatter.setPreferredSize(preferredSize);
    validate();
}
```

Megjegyzés: A 8-as verziótól kezdve a Java nyelv lambda kifejezéseket is képes kezelni (erre a feladatgyűjtemény feladataiban nem térünk ki, de nézzen utána, ha érdekli). Az esemény-figyelést ily módon kicsit tömörebben is meg lehet oldani:

```
        rdButton.addActionListener((ActionEvent ae) -> {
            int index = (((JRadioButton) ae.getSource()).getY()
                    - fuggolegesTav) / koztesTav;
            vezerlo.szavaz(index);
        });
```

A gombnyomások állítják be azt, hogy láthatóak legyenek-e a szavazatszámok, mégpedig ennek a metódusnak a segítségével:

```
private void szavazatSzamLathato(boolean lathato) {
    buttonGroup1.clearSelection();
    for (Component component : pnlHatter.getComponents()) {
        if(component instanceof JLabel)component.setVisible(lathato);
    }
}
```

A szavazást a vezérlő osztály oldja meg:

```
public void szavaz(int index) {
    Konyv konyv = beolvasottKonyvek.get(index);
    konyv.szavazatotKap();
    adatfrissites(konyv);
    konyvPanel.frissit(index, konyv.szavazatSzamSzoveg());
}

private void adatfrissites(Konyv konyv) {
    try {
        dao.update(konyv);
    } catch (Exception ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Azért kell a szöveget is átadnunk, mert a könyvpanel az eredeti könyveknek csak a másolatát kapta meg, vagyis ott nem változik a szavazatszám.

A `KonyvPanel` osztály `frissit()` metódusa:

```
public void frissit(int index, String szavazatSzamSzoveg) {
    int labelIndex = 2 * index + 1;
    if (pnlHatter.getComponents()[labelIndex] instanceof JLabel) {
        ((JLabel) pnlHatter.getComponents()[labelIndex]).
                                                setText(szavazatSzamSzoveg);
    }
}
```

Megjegyzések:

1. Az indexet azért számoltuk így, mert a panelen vegyesen szerepelnek rádiógombok és labelek.

2. A típuskényszerítés veszélyes dolog, nem szabad feltételvizsgálat nélkül alkalmaznunk, még akkor sem, ha itt tudjuk, hogy az adott indexen csak label lehet. (A rádiógomb

eseményfigyelésekor is alkalmaztunk típuskényszerítést, de ott valóban csak rádiógomb lehet a kényszerített példány, hiszen az az esemény forrása.)

3. Ahhoz, hogy tudjunk bánni az adatbázissal, a *Libraries* mappához hozzá kell adni a Derby kezeléséhez szükséges osztályokat (*JavaDb Driver*). Ez elég kellemetlen, annál is inkább, mivel most kódból hoztuk létre az adatbázist, és könnyen elfelejtkezünk erről a kötelezettségünkről. Sokkal kényelmesebb, és kellemesebb, ha a megoldáshoz Maven projektet használunk. Ha majd átvette ezt a témakört, térjen ide vissza, és alakítsa át ezt a projektet Maven projektté. NetBeansben ennyi az egész: hozzon létre egy új Maven projektet, másolja át bele az itteni csomagokat, és írja át a *pom.xml* fájlt a tanult módon. A projekt *src\java* mappájában a *main* mappa mellett hozzon létre egy *resources* nevezetű mappát (fájlkezelőben), és ide másolja be az eredeti projekt *adatok* nevű mappáját. Ha készen van, máris fut a projekt, nem kell külön vacakolni a Derby beállításával.

4. Hasonlóan lehet dolgozni más adatbázis-kezelővel is, csak értelemszerűen utána kell nézni, hogy abban hogyan lehet ellenőrizni egy tábla meglétét.