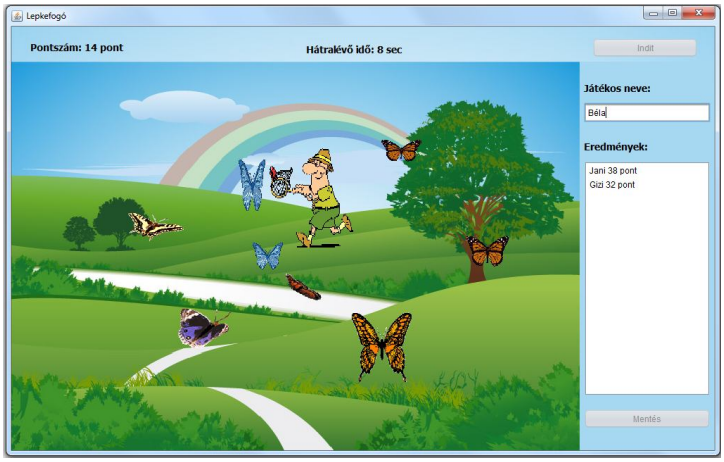
A pillangós feladat adatbázis része

Feladat:



Folytassuk a múltkori játékot, és írjuk meg az adatbázis-kezelő részt is!

Vagyis:

Induláskor adjuk meg a játékos nevét, és ha lejárt a játékideje, a listában jelenjen meg a neve és a pontszáma. Ha egy játékos többször is játszik, akkor válasszon a lehetőségek közül:

- csak a legutolsó eredménye jelenjen meg
- csak a legjobb
- esetleg a régi is és az új is, hogy könnyebben tudjon dönteni a mentésről

A listafelületen pontszám szerint csökkenően rendezve legyenek az adatok.

Még további folytatásként a listában lévő adatokat mentsük el egy adatbázisba, a következő induláskor az adatbázisból olvassa ki a játékosok adatait, és jelenítse is meg a listafelületen. Mentéskor értelemszerűen csak az újakat kell beszúrni, a korábban is létezőket pedig módosítani. Ehhez beágyazott Derby-t használunk majd Mavennel.

Megoldásrészletek

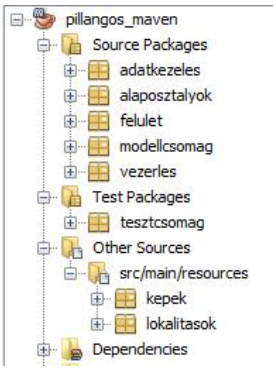
Bár legelső játékkor még nincs mit beolvasni, de mivel egyrészt az adatbázis-kezelő rész az igazán új ebben a feladatban, másrészt a program beolvasással kezdődik, ezért ennek tárgyalásával kezdem.

Most úgy oldjuk meg a feladatot, hogy

a) ne kelljen „kézzel” létrehozni az adatbázist

b) ne kelljen „kézzel” beállítani a drivert.

Az első célt a kód ügyes megírásával érjük el, a másodikat avval, hogy Maven projektet írunk. Kész Java projektet NetBeans-ben úgy lehet „mavenizálni”, hogy létrehozunk egy új Maven projektet, majd értelemszerűen bemásoljuk a korábbi Java projektet.



Ezek után annyi teendőnk van még, hogy megírjuk a pom.xml fájlt. Esetünkben ezeket a függőségeket kell bemásolni a generált fájlba:

```
<dependencies>

    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.14.1.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derbyclient</artifactId>
        <version>10.14.1.0</version>
    </dependency>

</dependencies>
```

Megjegyzés:
Ha megnézi a hazavitt projekt pom.xml fájlját, akkor láthatja, hogy abban még további két függőség is van. Ezek akkor generálódnak, amikor JUnit tesztfájlt hozunk létre.

Lássuk az adatbázis kezelést! Ezt a Dao tervezési minta alapján oldottam meg.
A Dao egy nagyon egyszerű tervezési minta, előírjuk benne az adatbázis-kezelés alapműveleteit (*CRUD – create*, *read*, *update*, *delete*). Ezeken kívül ide szokták még venni az összes adat listába való visszanyerését (`listAll()`).

A kiadott projekt interfésze most csak a listába gyűjtést, *create* és *update* műveletet írja elő, de a teljes Dao interfész még a törlést és a sima (azaz pl. az adott azonosítójú elemre vonatkozó) *read*-et is tartalmazza.

```
/**
 * Az interfész leírja, hogy milyen adatbáziskezelő műveleteket szeretnénk.
 *
 */
public interface Dao {

    public List<Jatekos> listAll() throws Exception;
    public void createJatekos(Jatekos jatekos) throws Exception;
    public void updateJatekos(Jatekos jatekos) throws Exception;

}
```

Az „igazi” Dao abban is eltér az itt bemutatott interfésztől, hogy ott nem konkrét típusú elemek kezeléséről van szó, hanem generikus módon van megfogalmazva, vagyis egy tetszőleges (vagy majdnem tetszőleges) <T> típusra, és a konkretizálás az implementáló osztály dolga.

Az interfészt megvalósító osztály:

```
public class JatekosDao implements Dao {

    private Connection kapcsolat;

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

    @Override
    public List<Jatekos> listAll() throws Exception {
        List<Jatekos> jatekosok = new ArrayList<>();

        if(kapcsolat != null){
            String sqlUtasitas = "SELECT * from JATEKOSOK";
            String nev;
            int pontszam;
            try(Statement utasitasObj = kapcsolat.createStatement();
                    ResultSet eredmenyHalmaz =
                                        utasitasObj.executeQuery(sqlUtasitas)){
                while(eredmenyHalmaz.next()){
                    nev = eredmenyHalmaz.getString("nev");
                    pontszam = eredmenyHalmaz.getInt("pontszam");
                    jatekosok.add(new Jatekos(nev, pontszam));
                }
            }
        }
        return jatekosok;
    }

    @Override
    public void createJatekos(Jatekos jatekos) throws Exception {
        if(kapcsolat != null){
            try(Statement utasitasObj = kapcsolat.createStatement()){
                String sqlUtasitas = "INSERT INTO APP.JATEKOSOK VALUES('"
                        + jatekos.getNev() + "', "
                        + jatekos.getPontSzam() + ")";
                utasitasObj.executeUpdate(sqlUtasitas);
            }
        }
    }

    @Override
    public void updateJatekos(Jatekos jatekos) throws Exception {
        if(kapcsolat != null){
            try(Statement utasitasObj = kapcsolat.createStatement()){
                String sqlUtasitas = "UPDATE APP.JATEKOSOK set pontszam = "
                        + jatekos.getPontSzam()
                        + " WHERE nev = '"
                        + jatekos.getNev() + "'";
                utasitasObj.executeUpdate(sqlUtasitas);
            }
        }
    }
}
```

Ahhoz, hogy ilyen módon tudjuk létrehozni a `Jatekos` példányokat, a `Jatekos` osztályban írnunk kell egy második konstruktort is.

Figyelje meg, hogy az *insert* műveletet is az `executeUpdate()` metódus segítségével oldjuk meg.

Néhány megjegyzés:
Elég kényelmetlen egy-egy összetettebb SQL utasítás megírása, hiszen állandóan figyelni kell a string műveletekre is, ezért most mutatok néhány egyszerűsítési lehetőséget:
1. A `Statement` helyett `PreparedStatement` típusú utasításobjektumot használhatunk, ekkor ugyanis paraméterezett SQL utasítást lehet kiadni, például így:

```
@Override
public void updateJatekos(Jatekos jatekos) throws Exception {
    if(kapcsolat != null){
        String sqlUtasitas =
                "UPDATE APP.JATEKOSOK set pontszam = ? WHERE nev = ?";
        try(PreparedStatement utasitasObj =
                                kapcsolat.prepareStatement(sqlUtasitas)){

            utasitasObj.setInt(1, jatekos.getPontSzam());
            utasitasObj.setString(2, jatekos.getNev());

            utasitasObj.executeUpdate();
        }
    }
}
```

2. A `String` osztály `format()` metódusával egyszerűbben meg tudjuk adni az SQL utasítást:

```
String sqlUtasitas =
      String.format("INSERT INTO APP.JATEKOSOK VALUES('%s',%d)",
      jatekos.getNev(),jatekos.getPontSzam());
```

Esetleg lehetne használni a `StringBuilder` vagy a `StringBuffer` osztályokat is, de talán ennél a kis feladatnál elég ennyi.
(https://stackoverflow.com/questions/2971315/string-stringbuffer-and-stringbuilder)

Elvileg még jobb megoldás lenne, ha a JDBC helyett a JPA megoldást választanánk, de gyakorlatilag ilyen kis feladatra nem érdemes a JPA-t használni, másrészt sok cég ragaszkodik még a JDBC használatához, ezért fontos annak megismerése is.

Az adatbázishoz az alkalmazás megnyitásakor kapcsolódunk, ekkor töltjük be a korábban elmentett eredményeket.

A `Vezerlo` osztály `adatBazisMegnyitas()` metódusa (a `Vezerlo` `beallitas()` metódusában hívjuk meg.):

Az adatbázist a `kapcsolodas()` metódusban hozzuk létre – feltéve persze, hogy még nem létezett. Vagyis ellenőrizhetjük. hogy létezik-e már a tábla, és ha nem, akkor létrehozzuk. Ha nem csak üres táblát akarunk létrehozni, akkor persze az egészet célszerű külön metódusban megoldani.

```
private void adatBazisMegnyitas() {
    try {
        dao = new JatekosDao(kapcsolodas());
        beolvasottJatekosok = dao.listAll();
        jatekosPanel.induloListabaIr(beolvasottJatekosok);
    } catch (Exception ex) {
        Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
    }
}

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

    Connection kapcsolat = DriverManager.getConnection(url);

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

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

        if (!rs.next()) {
            String sqlTablaKeszites =
              " CREATE TABLE APP.JATEKOSOK ( nev varchar(50), pontszam int)";
            utasitasObjektum.execute(sqlTablaKeszites);
        }
    }
    return kapcsolat;
}
```

A `JatekosPanel` hivatkozott metódusa:

```
private RendezhetoListModel<Jatekos> jatekosModell =
        new RendezhetoListModel<Jatekos>();

public void induloListabaIr(List<Jatekos> beolvasottJatekosok) {
    for (Jatekos jatekos : beolvasottJatekosok) {
        jatekosModell.addElement(jatekos, true);
    }
}
```

A játékosokat természetesen csak akkor lehet rendezhető modellbe tenni, ha implementálják a `Comparable` interfészt.

Néhány részlet az eredmény kiírásáról, mentéséről:

Ha vége van egy menetnek (lejárt a játékidő), akkor a játékos panelt megkérjük a játék végéhez tartozó teendőkre, ami:

```
    public void jatekVeg(Jatekos jatekos) {
        btnInditas.setEnabled(true);
        eredmenytIr(jatekos);
    }

    private void eredmenytIr(Jatekos jatekos) {
        if(jatekosModell.contains(jatekos)){
           jatekosModell.removeElement(jatekos);
        }
        jatekosModell.addElement(jatekos,true);
    }
```

Most a legutolsó eredményt mentettük, de próbálja meg megoldani, hogy a legjobb legyen elmentve. Persze, a `Jatekos` osztályt fel kell készíteni arra, hogy jól működjön a `contains()` metódus, vagyis ott generálni kell az `equals()` és `hashcode()` metódusokat (akkor tekintünk egyformának két játékost, ha azonos a nevük).

A mentés gomb hatása:

```
    private void btnMentesActionPerformed(java.awt.event.ActionEvent evt) {
        List<Jatekos> jatekosok = new ArrayList<>();
        for (int i = 0; i < jatekosModell.getSize(); i++) {
            jatekosok.add(jatekosModell.getElementAt(i));
        }
        vezerlo.adatMentes(jatekosok);
    }
```

```
    public void adatMentes(List<Jatekos> jatekosok) {
        for (Jatekos mentendoJatekos : jatekosok) {
            try {
                if (beolvasottJatekosok.contains(mentendoJatekos)) {
                    dao.updateJatekos(mentendoJatekos);
                } else {
                    dao.createJatekos(mentendoJatekos);
                    beolvasottJatekosok.add(jatekos);
                }
            } catch (Exception ex) {
                Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Megjegyzések:

1. Létrehozáskor azért adjuk hozzá a játékost a beolvasott játékosok listájához, mert ez a lista tartalmazza az adatbázisban lévő adatokat.

2. Úgy is meg lehetne oldani – ekkor egy, sőt két ciklussal kevesebb – hogy a játékosok modelljén végigfutva nem hozunk létre egy átadandó listát, hogy azon a vezérlő futhasson végig, hanem egyenként kérjük meg a vezérlőt arra, hogy mentse el a modell aktuális elemét. Ez szerintem egyszerűbb megoldás, de a gyakorlatok többségében a lista átadása mellett szavaztak. ☺

Gyakorlaton nem foglalkoztunk vele, szerintem most nem is lényeges, de elvileg le kellene zárnunk az adatbázissal való kapcsolatot is. Most ezt is megmutatom, hátha valamikor szüksége lehet rá. Ezt a frame lezáró gombjához kellene rendelni.

A frame `inditas()` metódusában:

```
this.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        vezerlo.lezar();
    }
});
```

A hivatkozott metódus:

```
public void lezar() {
    if (kapcsolat != null) {
        try {
            kapcsolat.close();
        } catch (SQLException ex) {
            Logger.getLogger(JatekosDao.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

(Ehhez persze a kapcsolatot nem lokálisan, hanem globálisan kellene deklarálni.)

Ha szeretnénk, hogy programunkat jar fájlból is lehessen futtatni (márpedig szeretnénk), akkor a pom.xml fájlt ki kell egészíteni evvel a build részlettel (a `</project>` elé):

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                    <manifest>
                        <mainClass>vezerles.FoFrame</mainClass>
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```