SZAKDOLGOZAT

Nagy László

DE IK

Matematikai és Informatikai Intézet

Debrecen, 2007.

Debreceni Egyetem Informatikai Kar

Egy szoftver fejlesztése Java Swing és HSQLDB környezetben

Készítette: Nagy László

III. programozó matematikus

Témavezető:

Dr. Juhász István egyetemi adjunktus

Debrecen, 2007.

Tartalomjegyzék

Bevezetés

Dolgozatom célja, hogy végigkísérje egy Angol Szótár és Oktatóprogram kifejlesztését. A fejlesztéshez a Java nyelvet fogjuk használni, a felület felépítéséhez a Java Swing csomag nyújt majd segítséget és JDBC-n keresztül HSQLDB adatbázisból fogjuk nyerni a szükséges adatokat. Dolgozatom fejezeteit az itt felsorolt témakörök köré szeretném felépíteni, és mindegyiket a szükséges mértékben bemutatva kívánok segítséget nyújtani azoknak akik valamikor hasonló témakörben szeretnék kipróbálni magukat. Mindazonáltal, hogy a megfelelő mértékben megpróbálom bemutatni a felsorolt eszközöket. Eme dokumentum feltételezi az olvasó bizonyos jártaságát a Java nyelv használatában. Miután megismerkedtünk valamelyest a használatba kerülő eszközökkel elkezdjük a dolgozat témáját képező program tervezését. Az ezután következő fázis az implementálás mely során saját megoldásomat fogom vázolni. A dolgozat lezárásának a program tesztelését és egy egyszerűbb felhasználói kézikönyv készítését szánom.

Azontúl, hogy a program elsődleges célja a demonstráció természetesen mint termék is megállja a helyét. A termék szóval persze nem az anyagi vonzatokra gondolok, hanem arra, hogy felhasználók kezébe kerülve értékes eszközként szolgálhat. Hogy kinek is? Mindazoknak akik idegennyelvvel (angol) kívánnak ismerkedni, és szókincsüket szeretnék gyarapítani. A program lehetővé teszi a szavak és mondatok összegyüjtését, majd ezek módszeres felkérdezését. A tanulást tovább segíti, hogy a szavakat témakörökbe tudjuk csoportosítani.

Következék a dolgozatom első fejezete, és ismerkedjünk meg az alkalmazott adatbázissal.

HSQLDB

A Hypersonic SQL egy nyílt forráskódú teljes mértékben java alapú adatbázis motor mely először 1998-ban jelent meg. Kizárólag memóriában tárolt motorként indult de a 2000-es évre és a 1.43-as verziót elérve már teljes értékű SQL motorrá vált.

2001–ben mikor a Hypersonic SQL projekt végetért számos fejlesztő akik ezt a szoftvert használták összefogtak és az Interneten keresztül létrehozták a HSQLDB fejlesztői csoportot. A csoport a HSQLDB (1.60) első verzióját 2001 áprilisában adta ki. Ez a verzió már támogatta a trigger-eket és egyéb hasznos fejlesztések is végbementek rajta. A következő verzió júliusban látott napvilágot 1.61-es verziószámmal.

2002 A növekvő érdeklődés a HSQLDB iránt újabb és újabb fejlesztéseket eredményezett. A fejlesztés folytatódott a következő 12 hónapban melynek gyümölcse egy rendkívül meg növekedett funkcionalitású engine volt 1.7.0– ás jelzéssel. A következő még ez év októberében a 1.7.1-es mely további fejlesztéseket tartalmazott és rengeteg régi hibát kijavítottak benne.

2003-2004 Az 1.7.2-es változaton már folyt munka mikor az 1.7.1 –t kiadták. Ez a folyamat egészen 2004 júliusáig tartott, mikor is végre hosszas béta és alfa tesztelések után kiadták az 1.7.2-es verziót. Aztán folytatták a hibák javítását minek következtében 10 újabb revízió látott napvilágot. Időközben Thomas Mueller – a Hypersonic SQL eredeti készítője – javaslatokat tett a BOOLEAN típus és a NULL feltételek kezelésére. Ennek eredményei már megjelentek a 1.7.3-as verzióban.

2004-2005 A 2004-es év második felében párhuzamosan készült a most már 1.8.0-nak nevezett verzió. Ebben nagy szerepe volt Ocke Janssen-nek aki az OpenOffice.org-al való integrációt segítette. Újraírtak rengeteg kódot ami elsősorban a perzisztenciát felügyelte és a memóriába való cache-selést tette lehetővé. Ez nagymértékben rugalmassá tette a HSQLDB-t és már nagy méretű táblák kezelésére is képes volt. Ekkor több új fejlesztés is megjelent párhuzamosan. Az 1.8.0 első változata júliusban jött ki. Az 1.8.0 elkészítése során a kód nagymértékben modularizálódott, felülvizsgáltak egy sor belső funkciót, némelyiket újra is írták.

2005-2007 A fejlesztés következő fázisa párhuzamosan koncentrált a belső és az interfész területekre. Néhány nagyobb komponenst teljesen újraírtak. Ennek következtében nem siették el az új verzió hanem egy jól kitesztelt alfa változat került kiadásra.

Most az 1.9.0 verzió közeledik a végső stádiumához. A következő fejlesztéseken dolgoznak rajta:

- Extensive coverage of SQL and JDBC features, aimed at full SQL 2003 Foundation and JDBC 4 compliance.
- New query processor
- Extensive support for several new SQL types
- Better integration with Open Source projects that use HSQLDB
- Improved logging and statistics
- J2EE features such as XA transactions, connection pooling, JMX, etc.

- Support for very large LOBs (10 MB - 2 GB)
- Support for very large result sets
- More extensive test frameworks

Telepítési és használati útmutató

A HSQLDB jar csomag a /lib könyvtárban foglal helyet és a következő komponenseket tartalmazza:

- HSQLDB RDBMS
- HSQLDB JDBC Driver
- Database Manager (Swing and AWT versions)
- Query Tool (AWT)
- SQL Tool (command line)

Az eszközök futtatása
Minden eszköz futtatható a hagyományos módon a Java osztályok futtatásával.

Például:

```
java -cp ../lib/hsqldb.jar org.hsqldb.util.DatabaseManager
```

A Hsqldb eszközök Main osztályai

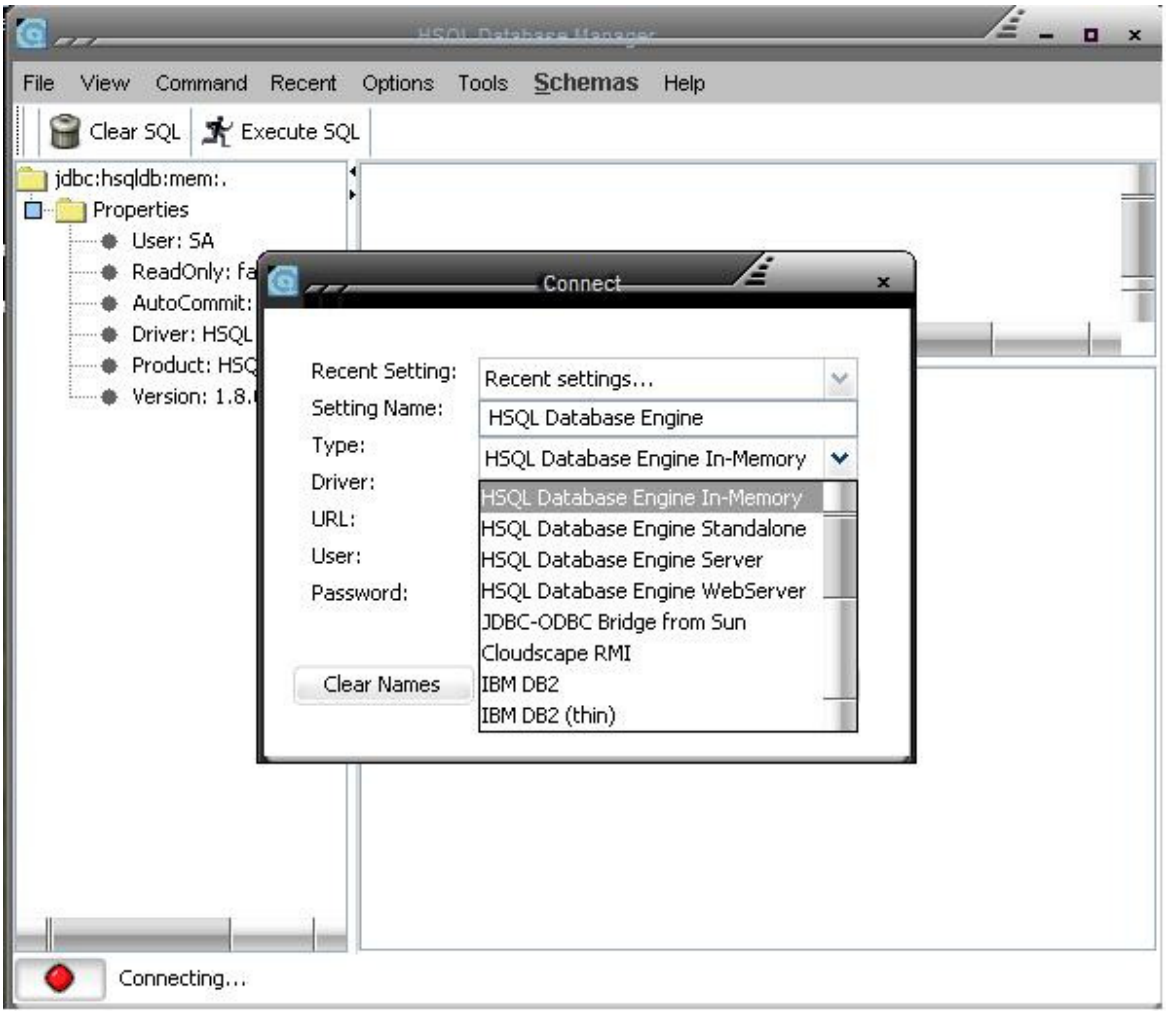
- org.hsqldb.util.DatabaseManager
- org.hsqldb.util.DatabaseManagerSwing
- org.hsqldb.util.Transfer
- org.hsqldb.util.QueryTool

- org.hsqldb.util.SqlTool

Néhány eszköz mint a Database Manager vagy az SQL Tool parancssorból is vezérelhető és paraméterezhető.

Database Manager

A csomagban kapjuk ezt a hasznos eszközt, mellyel könnyen hajthatunk végre SQL utasításokat az adatbázison. Indítása után csak meg kell adnunk az adatbázis szerver URL-t és a szerver módját és máris csatlakozhatunk hozzá.



Hsqldb fájlok

Mindegyik adatbázis 2-5 fájlt fog tartalmazni. A minta adatbázisban ezeket “test”-nek nevezi, és különböző kiterjesztéssel látja el:

- test.properties : az alapvető beállításokat tárolja az adatbázisról.
- test.script : tábla definíciókat és más adatbázis objektumokat ír le.
- test.log : az adatbázisban történő változás feljegyzésére szolgál.
- test.data : az adatbázisban tárolt adatok számára.
- test.backup : az utolsó konzisztens állapotot tárolja tömörített formában.

Szerver futási módok

Hsqldb Szerver

Ez az ajánlott módja az adatbázis szerver futtatásának, és egyúttal a leggyorsabb is. A parancs hasonló ahhoz ahogyan az eszközöket futtatjuk. A következő példában default beállításokkal indítunk egy szervert, mely esetben a fájlok neve “mydb.*” lesz.

```
java -cp ../lib/hsqldb.jar org.hsqldb.Server -database.0
file:mydb -dbname.0 xdb
```

Hsqldb Web Szerver

Ezt a módot akkor kell használni, hogyha a host és az adatbázis szerver között HTTP protocolon keresztül kívánunk kapcsolatot létesíteni. A futtatáshoz a fenti példában cseréljük az main osztályt a következőre:

```
org.hsqldb.WebServer
```

In-Process (Standalone) Mód

Ebben a módban az adatbázis motor a saját alkalmazásunk részeként, ugyanazon a JVM-en fut.

A legtöbb esetben ez gyorsabb működést eredményez, mivel nem kell konvertálni és hálózatra küldeni az adatokat. A legnagyobb hátránya is ez, mivel igy nem lehet kívülről csatlakozni az adatbázishoz. Így például nem tudjuk használni a Database Manager eszközünket.

Memory-Only Databases

E mód segítségével az egész adatbázisunkat a közvetlen hozzáférésű memóriában tárolhatjuk. Ebben az esetben semmilyen információ nem kerül merevlemezre. Az alkalmazások belső folyamatait vagy applet-ek működését segítheti egy ilyen adatbázis.

JDBC

A JDBC fejlődése

- JDBC 1.0

  A JDK 1.1-es verziójában jelent meg. Ez a kezdeti verzió már tartalmazza az adatbázis-eléréshez szükséges összes alapvető eszközrendszert.

- JDBC 2.0

  A JDK 1.2-es verziójában jelent meg. Két részre bomlott:

  - JDBC 2.1 alap API: az előző verzióhoz képest már tetszőleges sorrendben lehet feldolgozni az eredménytáblákat, ezeket már módosítani is lehet, nem csak olvasni; lehetőség nyílt a kötegelt parancsvégrehajtásra; kezelhetővé váltak az SQL3 adattípusok és a felhasználói SQL-típusok is.
  - JDBC 2.0 opcionális API: segítségével lehetővé vált adatforrások megadása a Java név- és katalógusszolgáltatáson keresztül; megvalósult az adatbázis-kapcsolatok újrafelhasználása, illetve azok globális tranzakciókban való részvétele; valamint megjelentek az önálló eredménytáblák.

- JDBC 3.0

  A JDK 1.4-es verziójában jelent meg. Az előző verzióhoz képest az alábbi újdonságokat tartalmazza:

  - Újrafelhasználható adatbázis-kapcsolatok esetén a kiadott parancsokat reprezentáló objektumok is újrafelhasználhatóvá váltak.
  - A tranzakciók tetszőleges köztes állapotig visszavonhatóak mentési pontok használatával.

- A PreparedStatement bemenő paramétereiről információkat lehet lekérdezni a ParameterMetaData interfész segítségével.
- Ha egy SQL-utasítás egyszerre több eredménytáblát is visszaad, azok akár egyszerre is feldolgozhatóak, nem csak egymás után.
- Szabályozható az eredménytáblák automatikus bezárása tranzakció véglegesítése esetén.
- Lekérdezhetővé vált a REF referenciával jelzett SQL struktúra értéke.
- A BLOB, CLOB, ARRAY és a REF SQL-típussal rendelkező adatok már nem csak olvashatóak, hanem írhatóak is.
- A JDBC már megérti a BOOLEAN és DATALINK SQL-típusokat is.

A JDBC használata

Ahhoz, hogy kliensként egy adatbázishoz kapcsolódhassunk és lekérdezhessünk adatokat, a következőket kell tenni:

1. Csomagokat kell importálni
2. Regisztrálni kell a JDBC meghajtó-programot
3. Kapcsolatot nyitni az adatbázis felé
4. Létre kell hozni a Statement objektumot
5. Végre kell hajtani a kérést, eredményként egy ResultSet objektumot kapunk
6. Fel kell dolgozni a ResultSet-et
7. Le kell zárni a ResultSet és a Statement objektumot
8. Végre kell hajtani a változtatásokat az adatbázison
9. El kell végezni véglegesítést

10.Le kell zárni a kapcsolatot.

Az első három pontban leírt tevékenységekhez HSQLDB driver specifikus információkat kell megadni, amely lehetővé teszi az adatbázishoz való kapcsolódást a JDBC API-n keresztül. A többi feladat esetén viszont a standard JDBC interfészeket alkalmazhatjuk Java alkalmazásainkban.

1. Csomagok importálása

Attól függően, hogy milyen HSQLDB JDBC meghajtó-programot használunk, a következő import utasításokat kell feltüntetni a programunk elején:

```
import java.sql.*;        - a standard JDBC csomagoknak
import java.math.*;       - a BigDecimal és BigInteger osztályoknak
import hsqldb.jdbc.*;     - az HSQLDB kiterjesztései JDBC-re
import hsqldb.sql.*;
```

2. A JDBC meghajtóprogramok regisztrálása

A DriverManager osztály kezeli a regisztrált meghajtó-programok listáját. Egy meghajtó-program rendszerint a betöltése után automatikusan, statikus incializátor segítségével regisztráltatja magát a DriverManager registerDriver() metódusával, ezért a felhasználónak csak a megfelelő meghajtó-program betöltéséről kell gondoskodnia. A betöltést a következő két módszer egyikével lehet megvalósítani:

- A meghajtó-program osztályának közvetlen betöltése a java.lang.Class osztály forName() nevű statikus metódusával, amely a paramétereként kapott nevű osztály dinamikus betöltését végzi el. Pl: Class.forName(”hsqldb.jdbc.HsqldbDriver”);

- A jdbc.drivers rendszerparaméter beállításával, ami a meghajtó-programok neveinek kettősponttal elválasztott listáját tartalmazza. A DriverManager osztály inicializálásakor automatikusan betölti ezen rendszerparaméterben felsorolt meghajtó-programokat. Ennek a módszernek az a hátránya, hogy appletek nem állíthatnak be rendszerparamétereket, és a DriverManager csak egyszer, az inicializáláskor veszi figyelembe a jdbc.drivers rendszerparaméter értékét. Ezért az első módszer használata javasolt.

3. Kapcsolat nyitása az adatbázis felé

A program és az adatbázis közötti kapcsolatot a Connection objektum reprezentálja. Egy program egyszerre több kapcsolatot is fenntarthat ugyanazon, vagy akár több különböző adatbázissal is. Egy kapcsolat a kiadott SQL-utasításokat és azok végrehajtásának eredményeit foglalja magában. A jelenlegi meghajtó-programoktól már elvárható, hogy azonos kapcsolaton belül kiadott több különböző SQL-utasítás feldolgozásának párhuzamosítása már ne a programozó, hanem a meghajtó-program feladata legyen.

Az adatbázis-kapcsolat felvételének menete a DriverManager osztály getConnection() metódusának meghívása, amelynek paramétere az elérni kívánt adatbázis-URL cím (és esetleg a felhasználói azonosító és jelszó). Ekkor a DriverManager megnézi, hogy a regisztrált meghajtó-programok közül melyik tudja a kapott adatbázis-URL-t feldolgozni, majd az első ilyen meghívja a saját connect() metódusát. A keresés sorrendje a meghajtó-programok regisztrációs sorrendjével egyezik meg, ahol a jdbc.drivers rendszerparaméterben megadott meghajtó-programok megelőznek minden közvetlenül regisztrált meghajtó-programot.

Az adatbázishoz való kapcsolódás egy olyan lépés, ahol HSQLDB JDBC driver specifikus információkat kell bejegyeznünk a getConnection() metódusban.

Ha egy kapcsolatban egy adatbázis nevét akarjuk specifikálni, akkor azt az alábbi módok valamelyikével tehetjük meg:

- egy Hsqldb Net kulcs–érték pár,
- a vékony driver esetén lehetséges a következő sztring: <hostnév>:<portszám>:<SID> (SID: ezzel a névvel azonosítja az Hsqldb a különböző adatbázispéldányokat),

4. Statement objektum létrehozása

Miután kapcsolódtunk az adatbázishoz, és létrehoztuk a Connection objektumot, a következő lépés a Statement objektum létrehozása lesz. A JDBC Connection objektum createStatement() metódusának visszatérési értéke a JDBC Statement osztályának egy objektuma. Az előző példák folytatása:

Statement stmt = conn.createStatement();

Ebben nincs semmiféle Hsqldb-specifikus dolog, megfelel a szabványos JDBC szintaktikának.

5. A kérés végrehajtása, és a ResultSet objektummal való visszatérés

Ha szeretnénk az adatbázisból adatokat lekérdezni, akkor a Statement objektumunk executeQuery() metódusát kell használnunk. Bemeneti paramétere egy SQL-utasítás, visszatérési értéke pedig egy JDBC ResultSet objektum. Az előbbi példát folytatván, most kérdezzük le az stmt nevű Statement objektumunktól az name nevű oszlopot az users táblából:

```
ResultSet rset = stmt.executeQuery („SELECT name FROM users”);
```

6. A ResultSet feldolgozása

Miután lefuttattuk a kérést, a next() metódus meghívásával az eredményhalmazt iterátorként járhatjuk be. Ennek a metódusnak a segítségével végig tudunk lépkedni sorról sorra az eredményen, azt is észrevéve, hogy mikor fogynak el az adatok.

Ahhoz, hogy végigmehessünk az eredményhalmazon, a megfelelő getXXX() metódusokat kell használnunk, ahol az XXX a megfelelő Java adattípusnak felel meg. Például:

```
while(rset.next())System.out.println(rset.getString(„name”));
```

7. A ResultSet és a Statement objektum lezárása

Mind a ResultSet, mind a Statement objektumot használatuk befejeztével explicit módon le kell lezárnunk. Ez az összes olyan objektumra vonatkozik, amelyeket az Hsqldb JDBC meghajtóprogramot használva hoztunk létre. A driverek nem zárják le ezeket az objektumokat, erre a close() metódust kell használnunk. Hogyha nincs lezárás, akkor egy idő után a memória megtelik, lassabb lesz a feldolgozás.

Például:

```
rset.close();
stmt.close();
```

8. A változtatások végrehajtása az adatbázison

Ahhoz, hogy az adatbázisba bekerüljenek az INSERT vagy UPDATE műveletek által okozott változások, tipikusan egy PreparedStatement objektumot kell létrehoznunk. Ez lehetőséget ad arra, hogy Statement objektumunkat különböző bemenő paraméterekkel futtathassuk. A JDBC Connnection objektum prepareStatement() metódusa által egy olyan Statement objektumot tudunk definiálni, amely kötött paraméterekkel rendelkezik.

A setXXX() metódusokat használjuk a PreparedStatement objektumok esetén az adatbázisban az általunk felviendő adatok beállítására.

Példa:

```
// új nevek beszúrása az EMP táblába
PreparedStatement p =
conn.prepareStatement („INSERT INTO users(username,
password, fullname, email) VALUES(?, ?, ?, ?)”);

ps.setString(1, „Bálint”);-- az 1. paraméter helyére kerül a
felhasználónév
ps.setString(2, ”picasso”);-- 2. paraméter a jelszó
ps.setString(3,”Nagy Bálint”);-- 3. a teljes név
ps.setString(4, „balint@insert.hu”);-- 4. egy e-mail cím
```

9. A változtatások véglegesítése

Alapértelmezés szerint, a DML utasítások (INSERT, UPDATE, DELETE) automatikusan véglegesítődnek, amint végrehajtásra kerülnek. Ez az ún. autocommit mód. A következő módon tiltható le:

```
conn.setAutoCommit(false);
```

Amennyiben ezt letiltjuk, nekünk kell gondoskodnunk a véglegesítésről, illetve a visszagörgetésről. Manuálisan kell megcsinálnunk a megfelelő Connection objektumon:

```
conn.commit();
```

vagy:

```
conn.rollback();
```

Fontos, hogy ha az autocommit mód le van tiltva, és lezárjuk a kapcsolatot a végrehajtott utasítások explicit véglegesítése nélkül zárjuk le, akkor végrehajtódik egy implicit COMMIT művelet.

Bármely DDL-utasítás, mint pl. a CREATE vagy ALTER után lefut egy implicit COMMIT utasítás. Ha az autocommit mód le van tiltva, akkor ez az implicit COMMIT nemcsak a DDL-utasításokat fogja véglegesíteni, hanem minden olyan folyamatban levő DML-utasítást is, amely még nem volt explicit módon véglegesítve.

10. A kapcsolat lezárása

A munka végeztével le kell zárnunk az adatbázis felé nyitott kapcsolatot is. Ezt a close() metódus segítségével tehetjük meg. Ez a metódus felszabadítja az adatbázis-kapcsolat által lefoglalt JDBC erőforrásokat, megszünteti az összes, a kapcsolaton keresztül kiadott utasítást és azok eredményeit reprezentáló objektumokat. Ez a metódus a kapcsolat-objektum megsemmisítésekor és bizonyos fatális hibák fellépésekor automatikusan is meghívódik. Ügyeljünk arra, hogy ha egy adatbázis-kapcsolatra nincs már szükségünk, feltétlenül zárjuk le a kapcsolatot, mert gyakran a háttérben felhalmozódó nyitva felejtett adatbázis-kapcsolatok vezetnek a programunk, vagy akár az adatbázisszerver összeomlásához. Ezért ajánlott az adatbázis-kapcsolat használatát egy try-blokkba beágyazni és a kapcsolatot a finally ágban lezárni.

A bemutatott példa kapcsolatának lezárása:

```
conn.close();
```

SQL-utasítások végrehajtása

Mivel a JDBC adatbázis-független API, ezért tudnia kell kezelni az adatbázis-kezelő rendszerek SQL megvalósításai közti apró különbségeket is. Ezen probléma megoldására a JDBC három módszert is biztosít:

- akármilyen szöveges utasítás (tehát nem csak SQL-utasítás!) átadható az adatbázisnak. Így egy adott adatbázistípus esetén maximálisan ki lehet használni annak minden specifikus lehetőségét is. A módszer hátránya, hogy a program elveszítheti hordozhatóságát, mivel nem biztos, hogy más adatbázis-kezelő is rendelkezik speciális funkciókkal.
- úgynevezett escape-szintaxis használata.
- az adatbázis-kezelő rendszer képességeinek lekérdezése a DatabaseMetaData interfész felhasználásával.

Az SQL-utasításokat a következő három interfész segítségével lehet végrehajtani:

- Statement

  Egyszerű SQL-utasítások végrehajtására használható.

- PreparedStatement

  Bemenő paraméterekkel is rendelkező SQL-utasítások ismételt végrehajtására használható.

- CallableStatement

  Ki- és bemenő paraméterekkel is rendelkező tárolt SQL-eljárások végrehajtására használható.

Megjegyzendő, hogy az SQL-utasítást lezáró, adatbázis-specifikus jelsorozatot (például pontosvessző) nem kell kitenni a végrehajtandó SQL-utasítás szövegének végén, ezt a meghajtó-program automatikusan megteszi helyettünk.

A Statement interfész

Statement létrehozása
Egy fennálló kapcsolatot reprezentáló Connection objektum createStatement metódusával hozható létre egy Statement objektum.

Statement végrehajtása

Egy Statement objektumot három metódusa segítségével is végre lehet hajtatni:

- executeQuery:

  A paraméterben megadott SQL-utasítást végrehajtja és egy annak eredményét reprezentáló eredménytábla (ResultSet) objektumot ad vissza. Lekérdező SQL utasítások (SELECT) végrehajtására használható.

- executeUpdate:

  A paraméterben megadott SQL-utasítást végrehajtja és a megváltoztatott adatbázistábla-sorok számát (update count) adja vissza. Adatmanipulációs (INSERT, UPDATE, DELETE) és adatdefiníciós (pl. CREATE/DROP TABLE) SQL-utasítások végrehajtására használható. Adatdefiníciós SQL-utasítások végrehajtása esetén a visszaadott érték mindig 0.

- execute:

  Ez a metódus is a paraméterében megadott SQL-utasítást hajtja végre, de az előző két végrehajtó metódus általánosításának tekinthető. Akkor lehet használni, ha az SQL-utasítás egyszerre többfajta eredményt is visszaadhat (pl. tárolt eljárások esetén), vagy nem ismert, hogy milyen típusú a

visszaadott eredmény (pl. ismeretlen SQL-utasítás dinamikus végrehajtásakor).

Az execute logikai visszatérési értéke igaz, ha az első visszaadott eredmény egy eredménytábla, különben hamis. Egy visszaadott eredménytáblát a getResultSet() metódussal lehet megkapni. Ez a metódus null-t ad vissza, ha a legutóbbi eredmény nem eredménytábla típusú, vagy ha már nincs több visszaadott eredmény. Adatmanipulációs utasítások esetén a megváltoztatott sorok számát a getUpdateCount() metódussal lehet megkapni. Ez a metódus –1-et ad vissza, ha a legutóbbi eredmény nem eredménytábla típusú, vagy ha már nincs több visszaadott eredmény. A következő visszakapott eredménykomponenst a getMoreResult() metódussal lehet megkapni. Ez a metódus logikai értéket ad vissza, melynek jelentése megegyezik az execute() visszatérési értékénél említettekkel.

Statement megszakítása

Mivel az utasítást végrehajtó programszál az SQL végrehajtás teljes időtartamára blokkolttá válik, ezért túl sokáig futó adatbázis-lekérdezések esetén felmerülhet az igény a végrehajtás megszakítására. Erre ad módot a cancel() metódus, amit ezek szerint egy másik programszálból kell meghívni. Sajnos ez a lehetőség erősen adatbázis- és meghajtó-program függő. Ha mindkét oldal támogatja ezt a lehetőséget, akkor a megszakított utasítást végrehajtó programszál kikerül blokkolt állapotából és SQLException fog kiváltódni.

Statement befejeződése

Az SQL utasítás végrehajtását akkor nevezzük befejezettnek, ha annak végrehajtása után az összes visszaadott eredménykomponens feldolgozásra került. Egy Statement objektum végrehajtásakor ugyanazon objektum esetleges korábbi végrehajtása által visszakapott eredménytábla automatikusan lezárul. Ez azt jelenti, hogy egy Statement újbóli végrehajtása előtt mindig teljesen fel kell

dolgozni az előző végrehajtásából származó esetleges eredménytáblá(ka)t. Figyeljünk arra, hogy minden utasításobjektumot a végrehajtása és esetleges eredményei feldolgozása után mindig azonnal zárjunk le annak close() metódusával, mert a háttérben felgyülemlő nyitva maradt Statement objektumok nagyon hamar elfogyasztják szabad erőforrásainkat, mivel általában igen memória- és erőforrás-igényesek. Ennek legegyszerűbb módja, ha az utasítás végrehajtása és az esetlegesen visszaadott eredményeinek feldolgozása egy try-blokkba beágyazva történik, melynek finally ágában gondoskodhatunk az utasításobjektum lezárásáról. Mivel rendszerint az adatbázis-kapcsolatra is csak addig van szükségünk, míg végre nem hajtottunk pár SQL-utasítást, ezért érdemes a Statement lezárása után mindjárt a Connection-t is lezárni.

A PreparedStatement interfész

Ez az interfész a Statement interfész kiterjesztettje, két különbség van közöttük:

- az interfészt megvalósító objektum egy adott SQL-utasítást tárol, méghozzá előfordított formában.
- az SQL-utasítás tartalmazhat bemenő paramétereket is, melyeket az SQL-utasításon belül kérdőjelek jelölik, mivel értékük az utasítás létrehozásakor még nem ismert. Az utasítás végrehajtása előtt minden bemenő paraméternek értéket kell adni a megfelelő beállító metódusok valamelyikével.

Mivel ez az SQL utasítástípus előfordított, ezért gyorsabb a végrehajtás, mint a Statement objektumok esetében.

PreparedStatement létrehozása

Egy fennálló kapcsolatot reprezentáló Connection objektum prepareStatement() metódusával hozható létre egy PreparedStatement interfészt megvalósító objektum. A végrehajtandó SQL-utasítást az

objektumot létrehozó, nem pedig az azt végrehajtó valamelyik metódusnak kell megadni. Például:
prepareStatement(”UPDATE tábla SET oszlop1 = ? WHERE oszlop2 = ?”)

PreparedStatement végrehajtása
A végrehajtásra ugyanaz a három metódus használható, mint a Statement esetében. Annyi a különbség, hogy ezeknek a metódusoknak nem kell paramétert megadni, mivel a végrehajtáskor már ismert az elvégzendő SQL-utasítás. Ezért a végrehajtható metódusokat paraméter nélkül kell meghívni, különben SQLException-t fognak kiváltani. Végrehajtás előtt minden bemenő paraméternek be kell állítani az aktuális értékét.

A PreparedStatement végrehajtásának sebességbeli előnye akkor válik nyilvánvalóvá, ha ugyanazt az SQL-utasítást (esetleg más paraméterekkel) többször is el kell végezni egymás után. A kapcsolat alapértelmezett automatikus véglegesítési módja esetén ugyanazon SQL-utasítást csak akkor lehet többször is végrehajtani egymás után, ha az adatbázis-kezelő véglegesítéskor nem zárja le a nyitott SQL-utasításokat. Ettől eltekintve nem lesz semmilyen sebességbeli különbség egy előfordított és egy sima SQL utasítás végrehajtása között, sőt az előfordítás miatt egyszeri használatkor még lassabb is lehet a végrehajtás.

Bemenő paraméterek típusának lekérdezése
A PreparedStatement objektum paramétereiről a getParameterMetaData() metódussal szerezhetünk információkat. A visszaadott ParameterMetaData objektumtól lekérdezhető a paraméterek száma (getParameterCount()), valamint megtudható adott sorszámú paraméter módja (getParameterMode: parameterModeIn a bemenő, parameterModeOut a kimenő, és parameterModeInOut a be/kimenő paramétereknél), osztályneve (getParameterClassName), SQL-típusa (getParameterType,

getParameterTypeName), értékes jegyeinek száma (getPrecision, getScale), valamint hogy megengedett-e a paraméter üresen hagyása (isNullable).

Bemenő paraméterek értékének megadása

Egy bemenő paraméter értékét a set<típusnév> alakú metódusokkal lehet beállítani. Ezen metódusok első argumentuma mindig a paraméter sorszáma (1-től kezdve), második argumentuma pedig a beállítandó értéket adja meg. A beállított érték mindaddig megmarad, míg a paraméterek értékét újra be nem állítjuk, vagy töröljük azokat a clearParameters() metódussal. Bemenő paraméternek NULL értéket a setNull() metódussal lehet beállítani.

A CallableStatement interfész

Ez az interfész a PreparedStatement interfész kiterjesztettje. Tárolt PL/SQL eljárások hívására kell használni. Míg a tárolt eljárást maga az adatbázis tartalmazza, addig egy CallableStatement objektum csak ezen eljárást meghívó utasításból áll. Egy ilyen tárolt eljárás hívása a bemenő paramétereken kívül kimenő paramétereket is használhat. Ezek jelölése ugyanúgy kérdőjellel történik, akárcsak a bemenő paramétereké. Egy paraméter egyszerre lehet kimenő és bemenő paraméter is.

CallableStatement létrehozása

Egy fennálló kapcsolatot reprezentáló Connection objektum prepareCall() metódusával hozható létre egy CallableStatement interfészt megvalósító objektum. A végrehajtandó PL/SQL tárolt eljárás hívását az objektumot létrehozó, nem pedig az azt végrehajtó valamelyik metódusának kell megadni.

Maga az eljáráshívás escape-szintaxis segítségével történik, a következő két forma valamelyikével:

- {call <eljárásnév> (?, ?, …)}
  visszatérési érték nélküli tárolt eljárás meghívása. Argumentumok nélküli eljárás esetén a zárójeles paraméterlista elhagyható.
- {? = call <eljárásnév> (?, ?, …)}
  eredményt visszaadó tárolt eljárás hívása. Argumentumok nélküli eljárás esetén a zárójeles paraméterlista elhagyható.

CallableStatement végrehajtása

A végrehajtás ugyanúgy történik, mint egy PreparedStatement esetén. Végrehajtás előtt azonban nem csak a bemenő paramétereknek kell beállítani az aktuális értékét, hanem minden kimenő paraméternek is meg kell adni a típusát a registerOutParameter() metódus segítségével.

Kimenő paraméterek értékeinek lekérdezése

Egy kimenő paraméter értékét a get<típusnév> alakú metódusokkal lehet lekérdezni. Ezek első argumentuma mindig a paraméter sorszáma (1-től kezdve). Javasolt, hogy a maximális hordozhatóság érdekében a kimenő paraméterek értékét csak minden visszaadott eredménykomponens feldolgozása után kérdezzük le, illetve az értékek feldolgozása sorban és csak egyszer történjék meg.

Eredménytáblák kezelése

SQL lekérdezések eredményét mindig egy eredménytábla (ResultSet) objektum reprezentálja. Ez az interfész az eredménytábla soronként történő elérését, az aktuális sor megfelelő oszlopában levő érték kezelését teszi lehetővé. A JDBC már lehetővé teszi az eredménytáblák sorainak tetszőleges sorrendben történő

feldolgozását, sőt az eredményeket nemcsak beolvasni, hanem akár módosítani is lehet.

Grafikus felhasználó felület

Graphical User Interface ,GUI

Egy program futása során rendszerint valamilyen módon biztosítani kell a program és a felhasználó között kommunikációt. Legegyszerűbb esetben elegendő, ha csak az indításkor adunk meg paramétereket és azután a programunk már nem igényel további beavatkozást. Tipikusan ilyen feladat lehet különböző számítások elvégeztetése vagy fájlok feldolgozása/előállítása ekkor a java.io csomag felhasználásával kényelmesen megoldható a szükséges adatok átadása és átvétele. Más esetekben pedig egyszerűen nincs is rá igény, hogy a program emberi felhasználókkal kommunikáljon.

A programok nagy részével azonban a felhasználó interaktív módon kerül kapcsolatba, ami azt jelenti, hogy a felhasználó és a program között (inter-) tulajdonképpen egy élő párbeszéd alakult ki: a felhasználó adatokat közöl a programmal, majd utasításokat ad ezen adatokkal történő műveletek elvégzésére, melyek eredményét a program visszajelzi, esetleg további adatokat kér be, vagy választási lehetőségeket kínál fel. Ezen működési megvalósítható karakteres kommunikációval is, ami valóban hasonlít egy párbeszéd lefolyására, de leggyakrabban mégis egy grafikus felhasználói felületen keresztül történik.

A grafikus felületek nagyban gyorsíthatják és egyszerűsíthetik a számítógép kezelését a felhasználók számára, mert lehetővé teszik a kommunikációban annak a ténynek a kihasználását, hogy az emberi agy a grafikákat és jól megkülönböztethető, jellemző szimbólumokat sokkal gyorsabban képes azonosítani és értelmezni, mint a szöveges, verbális információkat.

A grafikus felhasználói felület tehát lehetőséget biztosít adatok bevitelére, műveletek elvégeztetésére és eredmények kijelzésére. Megjelenési formája a megjelenő eszközön képszerű (azaz grafikus), ahol a felület egy-egy vizuálisan jól behatárolható része felelős ezen részfeladatok valamelyikének ellátásáért. Ezeket nevezzük grafikus komponenseknek. A fő feladatuk alapján a következőképp csoportosíthatók:

- beviteli komponens : segítségükkel adatokat adhatunk át a programnak
- vezérlő komponens : a program futását lehet velük vezérelni
- megjelenítő komponens: információt jelenítenek meg a felhasználó felé.

A felhasználó felület életciklusa

Objektumorientált nyelvek esetén természetesen a felhasználói felületet is objektumok reprezentálják. A grafikus komponenseknek egy objektum feleltethető meg, ezen objektumokból épül fel a teljes felület. Ennek megfelelően a felhasználói felület életciklusa a következő három szakaszra osztható fel:

Felület felépítése: ilyenkor kell létrehozni a megfelelő objektumokat, beállítani azok jellemzőit és egymás közti kapcsolatait.

Felület használata : ez az életciklus leghosszabb és a felhasználó szempontjából legfontosabb szakasza, hiszen ekkor tud a felület segítségével a felhasználó programunkkal kommunikálni.

Felület bezárása: a kommunikáció lezárása után gondoskodni kell a felületet felépítő objektumok által lefoglalt erőforrások felszabadításáról.

Eseményvezérelt programozás

Grafikus felhasználói felületek esetén a program és a felhasználó közti kommunikáció általában már nem hasonlítható egy egyenes lefolyású párbeszédhez. A felületnek ugyanis elsődleges feladata a felhasználó adatbevitelének megkönnyítése, aminek sorrendje, időbeli lefolyása rendszerint teljesen kötetlen. Ezért nem alkalmazható a szekvenciális programozási technika, ahol is a program egy meghatározott pontján ha adatbevitelre van szükség , a program rákérdez a megfelelő értékre és mindaddig vár, míg a felhasználó azt meg nem adja.

Grafikus könyvtárak

A java.awt és a java.awt.event csomag a grafikus felhasználói felület megvalósítására szolgál. Az ebben a csomagban definiált osztályok olyan komponenseket reprezentálnak, amelyekhez a java virtuális gépet futtató operációs rendszer ablakozó rendszerében valódi objektumok (nyomógombok, editorok...) tartoznak. Létezik egy pehelysúlyú komponenskönyvtár is, a Swing, az ebben definiált komponensek úgy jelennek meg a képernyőn, hogy kirajzolják magukat. Az AWT csomag komponenseinek kinézete a konkrét ablakozó rendszertől függ, a Swing-es komponensek minden rendszerben ugyanúgy néznek ki.

AWT

Az **Abstract Windowing Toolkit** (absztrakt ablak-eszközkészlet) vagy **AWT** a Java programozási nyelv ablakkezelésre és grafikus felhasználói felületek létrehozására szolgáló komponensgyűjteménye. Az AWT a JFC része, a Java osztályhierarchiában a jawa.awt helyen található.

Az AWT az operációs rendszer magas szintű, grafikus komponenseket megjelenítő szubrutinjait használja fel a saját komponenseinek megjelenítéséhez, ezért más Java osztályoktól eltérően nem nyújt platform-független absztrakciós réteget: az AWT-ben írt alkalmazások megjelenése különböző operációs rendszereken nagyon eltérő lehet. Emiatt kezdetben, amikor még nem volt alternatívája, az AWT-t a Java egyik leggyengébb pontjának tartották.

A JDK 2-es verziójának megjelenésével az AWT-t nagyrészt kiszorította a Swing, egy részben az AWT-re épülő, fejlettebb komponensgyűjtemény, ami alacsonyszintű szubrutinok segítségével maga rajzolja meg a grafikus komponenseit, így a megjelenése független az operációs rendszertől (emellett több komponenst is tartalmaz, mint az AWT).

Grafikus komponensek

Egy grafikus felhasználói felület grafikus komponensekből épül fel, melyek megjelenítése a felület egy vizuálisan jól behatárolható részé történik. A komponenst egy objektum reprezentálja, ezek közös őse a java.awt csomagban található absztrakt Component osztály. A közös ősosztály miatt minden komponens a következő tulajdonságokkal rendelkezik:

- a komponens az őt érdeklő eseményeket az enableEvents és disableEvents protected metódusokkal jelzi z AWT felé. Ezen metódusok a

figyelni kívánt, illetve a többé már nem érdekes eseményeket egy eseménymaszk formájában várják paraméterül. A komponens egy adott típusú AWT eseményt akkor fog megkapni, ha annak figyelését jelezte az enableEvents segítségével, vagy az eseményhez regisztrálta már magát legalább egy eseményfigyelő.

- A túl sok felesleges eseményobjektum elkerülése végett egy esemény bekövetkeztekor ha már várakozik egy ugyan olyan típusú esemény ugyanazon forrásból, akkor az AWT meghívja a forráskomponens coalesceEvents protected metódusát, ami a már várakozó és az újonnan bekövetkezett esemény esetleges összevonásáért felelős. Például egérmozgatás esemény esetén elég csak a legutolsó egérpozícióról eseményt küldeni.

- Az AWT a komponenst annak dispatchEvent metódusának meghívásával értesíti egy figyelt esemény bekövetkeztéről. Tehát ez a metódus a felelős a komponens eseményekre történő reagálásának megvalósításáért. Rendszerint itt történik a figyelők értesítése a processEvent protected metódus segítségével, amely az esemény típusa alapján meghívja a megfelelő processXXXEvent protected metódust.

- A komponens összes regisztrált figyelőjét adott eseménytípushoz a getListeners metódussal lehet lekérdezni.

- A komponens vizuális alaptulajdonságainak megváltozásakor ComponentEvent esemény generálódik, amely az aktuálisan megváltozott komponenst (getComponent) tartja nyilván. Ezen esemény ComponentListener figyelőinek (melyek a komponens méretváltozásáról (componentResized), elmozdulásáról (componentMoved) és láthatósága

megváltozásáról (componentHidden, componentShown) kapnak értesítést) listáját az add/removeComponentListener metódusokkal kezelhetjük.

- A komponens billentyűeseményei figyelőinek listáját az add/removeKeyListener metódusokkal kezelhetjük.
- A komponens alap egéreseményeit figyelők listáját az add/removeMouseListenter, az egérelmozdulás figyelők listáját pedig az add/removeMouseMotionListener metódusokkal kezelhetjük.
- Egéresemények feldolgozásakor az érintett komponenst az AWT az egéresemény bekövetkeztének koordinátái alapján keresi meg. Ehhez minden komponens implementálja a contains metódust, amely a paramétereként kapott koordinátájú pontról eldönti, hogy az a komponenshez tartozik-e.
- Az egérkurzor alakját a komponens területén belül a get/setCursor metódusokkal lehet lekérdezni/beállítani.
- A komponens reagálását a felhasználó által kiváltott eseményekre a set/isEnabled metódusokkal lehet szabályozni/lekérdezni. Ha egy komponens reagálását letiltottuk, akkor azt rendszerint a megjelenítés “elszürkülése” is jelzi.
- A komponens grafikus megjelenítése annak paint metódusában történik, ami a komponens megjelenítése frissítését végző update metódus megvalósításának utolsó lépésenként kerül végrehajtásra. Ezen metódust pedig az AWT automatikusan akkor hívja meg, ha a komponenst (újra) ki kell rajzolni a megjelenítő eszközre, de a mi magunk is kezdeményezhetjük az újrarajzoltatást a repaint metódus segítségével. A paint és update metódusok paramétereként megkapják az aktuális rajzolási környezetet

reprezentáló Graphics objektumot. Alapértelmezés szerint az update a komponens teljes megjelenítési területét törli a beállított színeknek megfelelően, mielőtt a paint megkapná a vezérlést.

- A komponens bármikor létrehozhat egy rajzolási környezetet a getGraphics metódusával. Ha a komponens még nincs a megjeleníthető állapotban, akkor null lesz a visszaadott érték.

- A komponens előtér és háttérszínét a get/setForground és get/setBackground, szöveg megjelenítéséhez használt betűtípust pedig a get/setFont metódusaival lehet lekérdezni/beállítani. A komponens betűkészletének jellemzőit a getFontMetrics metódus adja vissza.

- A getLocationOnScreen metódus segítségével lekérdezhetőek a komponens bal felső sarkának képernyő-koordinátái, a get/setSize metódusokkal pedig a komponens mérete kérdezhető le/állítható be. Ehhez használható a getHeight és a getWidth metódus is, melyek komponensnek csak a magasságát és a szélességét adják vissza képpontokban mérve.

- A különböző grafikus felületek felépítését segítő programok miatt minden komponens rendelkezhet egy tetszőleges névvel, amit a get/setName metódusokkal lehet lekérdezni/beállítani. Ez a lehetőség csak a komponens könnyebb azonosíthatóságát szolgálja, az AWT a komponens ezen tulajdonságát nem használja semmire.

- Minden komponenshez rendelhető egy java.util.Locale objektum a get/SetLocale metóduseokkal, ezáltal nyelvterületfüggően végezhető el a komponens megjelenítése.

- A komponenshez tartozó Toolkit objektumot a getToolkit metódussal lehet lekérdezni.

- Minden komponens megvalósítja a java.awt.image.ImageObserver interfészt, ezért képelőállítás menetének felügyeletére képes. Adott kép előállításának megkezdését a prepareImage metódussal lehet explicit kérni. A képelőállítás állapotát ekkor a checkImage metódus segítségével lehet ellenőrizni, s a java.awt.image.ImageObserver interfészben definiált konstansok bitenként össszeVAGY-olt értéke lesz a visszaadott érték. Adott méretű üres képet létrehozni a createImage metódussal lehet, amit rendszerint az offscreen technika megvalósításához szükséges memóriabeli kép-puffer létrehozásához használnak. Ennek lényege, hogy a megjelenítendő képet előbb a memóriában “rajzoljuk” meg, majd a kész képet egyszerre jelenítjük meg a képernyőn.

Konténerek és Elrendezési stratégiák

A felhasználói felületet felépítő grafikus komponensek nagy számossága miatt a könnyebb kezelhetőség érdekében szükségessé vált azok csoportokba történő összevonása. Ez egy hierarchikus tartalmazási struktúra bevezetését jelentett, ami egy újfajta grafikus komponens, az úgy nevezett konténer komponensek segítségével valósítható meg. A konténer komponensek legfőbb feladata más grafikus komponensek összefogása egy csoporttá, ahol a csoportnak maga a konténer komponens feleltethető meg. A csoport elemeit alkotó grafikus komponenseket a konténer elemeinek, a konténerelemek és a konténer kapcsolatát pedig tartalmazásnak nevezzük (ez a kapcsolat egy tartalmazási relációt definiál) . Ennek előnye, hogy például elég a konténer elemeit csak a konténeren belül pozícionálni, ezután azok tényleges elhelyezkedése a felületen egyszerűen a konténer pozíciójának változásával összehangoltan fog módosulni.

A konténerelemek megjelenítésének különféle elrendezési lehetőségét nevezzük elrendezési stratégiának. Egy ilyen stratégiát a LayoutManager, illetve annak kiterjesztettje a LayoutManager2 interfész ír le, az ezt megvalósító objektumot pedig a konténer get/setLayout metódusaival lehet lekérdezni/beállítani.

- A CardLayout elrendezési stratégia egyszerre csak komponenst tesz láthatóvá, a többi komponens úgymond a látható komponens alatt takarásban van, azért azok nem látszanak.

- A FlowLayout elrendezési stratégia a komponenseket sorokba szervezi, minden komponenst annak optimális nagyságára méretezve.

- A BorderLayout elrendezési stratégia egyszerre legfeljebb öt komponenst tud kezelni, melyek igazítása a konténer széleihez, valamint a fennmaradó helyre történik. Ezen a igazítást a szöveges típusú elrendezési jellemző határozza meg, melynek értékei a BorderLayout konstansai lehetnek.

- A GridLayout elrendezési stratégia a rendelkezésre álló helyet oszlopokra és sorokra, ezáltal egyenlő nagyságú cellákra osztja fel. Minden komponens egy ilyen cellát foglal el, tehát ezen stratégia szerint minden komponens egyforma méretű lesz.

- A GridBagLayout elrendezési stratégia a rendelkezésre álló helyet szintén oszlopokra és sorokra osztja fel, viszont ezen oszlopok és sorok különböző szélességűek és magasságúak lehetnek, a komponensek pedig egyszerre több cellahelyet is elfoglalhatnak.

AWT és SWING felületelemek

- **A Canvas** a legegyszerűbb komponensek egyike, amely önmagában nem képes semmire, nem is rendelkezik saját megjelenítéssel, viszont a paint metódusát felüldefiniálva lehet rajzolni a felületére, valamint a lehetővé teszi a területén belül fellépő felhasználó által kiváltott események figyelését.

- **Címke** feladata csak egy szöveg megjelenítése és semmilyen felhasználó által kiváltott eseményre sem reagál.

- **Nyomógomb**-ot a Button osztály reprezentálja. Megnyomásakor ActionEvent esemény generálódik, amely az AWTEvent leszármazottjaként az esemény bekövetkeztekor lenyomva tartott módosító billentyűket és az esemény forrásához hozzárendelt szöveges parancskódot tartja nyilván.

- **A Rádiógomb** a Checkbox osztály által reprezentált kiválasztható gomb vizuálisan jól megkülönböztethető be és kikapcsolt állapota segítségével egy logikai érték megjelenítését teszi lehetővé. Feliratának megadása a konstruktorban, vagy a get/setLabel metódussal történhet.

- **Szövegmező** ősosztálya a TextComponent, segítségével szabályozható a szöveg szerkeszthetősége, lekérdezhető és megváltoztatható a szöveg tartalma, valamint lehetővé teszi adott szövegrész kiválasztását.

- **Szöveges listák** görgethető megjelenítését a List osztály végzi. A lista elemeit az add, replace és remove metódusokkal lehet hozzávenni, felülírni és törölni.

- **Legördíthető listát** reprezentáló Choice osztály egy olyan szöveglistát valósít meg, melynek mindig csak egy eleme lehet kiválasztva, a listából

pedig csak ezen kiválasztott elem látható. A teljes lista csak akkor érhető el, ha azt a kiválasztás idejére legörgetjük.

- **Menürendszert** megvalósító komponensek az AWT komponenshierarchiától elkülönülve egy önálló hierarchiát alkotnak, ennek az absztrakt MenuComponent osztály a csúcspontja. A menük különlegessége, hogy gyorsítóbillentyű rendelhető hozzájuk. A menüpontokat a MenuItem osztály reprezentálja.

SWING rövid ismertetés

A Swing elnevezés a JFC (Java Foundation Classes) grafikus komponenseit fejlesztő project kódneve volt. A JFC a Sun ipari Java alkalmazások fejlesztését elősegítő programkönyvtár gyűjteménye, amely a Netscape IFC (Internet Foundation Classes) és az IBM Taligent részlegének együttműködésével jött létre. A Swing olyan grafikus komponensek gyűjteménye, melyek az AWT-re, annak eseménykezelő modelljére és pehelysúlyú komponens lehetőségére épülnek - ezért tartottam fontosnak ismertetni az AWT komponenseit.

Ennek legnagyobb előnye, hogy a Swing teljesen Javában íródhatott, így független lett a grafikus felületet megjelenítő operációs rendszertől.

A Swing komponensek az AWT komponensei továbbfejlesztésének tekinthetőek. Erre az is utal, hogy a Swing tartalmazza az AWT összes nehézsúlyú komponensének pehelysúlyú változatát (kivétel a Canvas), és ezek nevei az AWT komponensek nevéből a név elé írt nagy J betűvel képezhetők. Tehát a Swing grafikus felületek létrehozására és használatára mindaz igaz, amit az AWT ismertetésénél már leírtam, felmerülő különbségeket pedig az implementáció tárgyalásánál fogom kifejteni.

Tervezés

Mivel ez a program elsősorban demonstrációs célt szolgál majd, ezt már a tervezésnél figyelembe kell venni. Ezt szem előtt tartva megpróbálom majd kiaknázni a használatba kerülő API-k lehetőségeit. A SWING esetén minél több felületelemből építkezem majd, a JDBC esetén pedig megvalósítok számos funkciót melyek kihasználják az API-t.

Már a tervezés során fontos arra törekedni, hogy elkülönüljön egymástól a felület (GUI) az adatkezelés és a logika hármasa. Az implementáció során ezeket külön csomagokba is fogom rendezni. Ez a látásmód többek között arra is jó lesz, hogy anélkül tudjuk könnyen módosítani a program felhasználói interfészét, hogy esetleg az adatszerkezethez, vagy a logikához kéne hozzányúlni.

Adatbázis tervezése

Az adatbázis tervezését érdemes körültekintően végezni, hogy megkönnyítsük a későbbi lekérdezéseket. Néhány tábla esetén talán nem létszükséglet, de 5-10 vagy akár több tábla esetén mindenképpen ajánlott valamilyen tervezőprogram használata, hogy előre lássuk a hibákat, és elkerüljük a félretervezést.

Mivel az adatbázisunk egyedeket és az ő kapcsolataikat hivatott absztrahálni, ezért tervezésnél érdemes ezt a két dolgot külön szemlélni. Az egyedeink jelen esetben táblák lesznek, melyek leírják az ők tulajdonságaikat.

Adatbázis táblák

Users tábla

Az a tábla ahol a felhasználókat fogjuk tárolni. Különösebb authentikációs szerepe nem lesz, csak arra használjuk, hogy megkülönböztessük az egyes felhasználókat, és a felkérdezések során elért eredményeiket.

Words tábla

A szavakat fogjuk itt tárolni. Azonkívül, hogy eltároljuk az adott szó több jelentését is (most csak angol és magyar, de ez akár bővíthető is lenne), eltárolunk még egy plusz információt is mely a szó egyedi azonosítója, elsődleges kulcsa lesz.

Lists tábla

Ez a tábla az adatbázisban tárolt szavak részhalmazainak azonosítására szolgál majd. Ily módon témaköröket definiálhat a felhasználó. Ezeket majd bővítheti vagy akár törölheti, anélkül, hogy a szavak adatbázisából törlődnének.

Sentences tábla

Ez a tábla nem rendelkezik kapcsolatokkal. A szavakhoz hasonlóan itt angol-magyar mondat párokat fogunk tárolni.

Kapcsolatok

Mivel a felhasználói listák felhasználókhoz tartoznak, szükség lesz egy kapcsolat kiépítésére a Users tábla és a Lists tábla között. Ezt én úgy kívánom megoldani, hogy a Lists táblában elhelyezek egy külső kulcsot, mely a Users tábla elsődleges kulcsára fog referálni. Így úgymond a Listák tudni fogják magukról mely felhasználóhoz tartoznak.

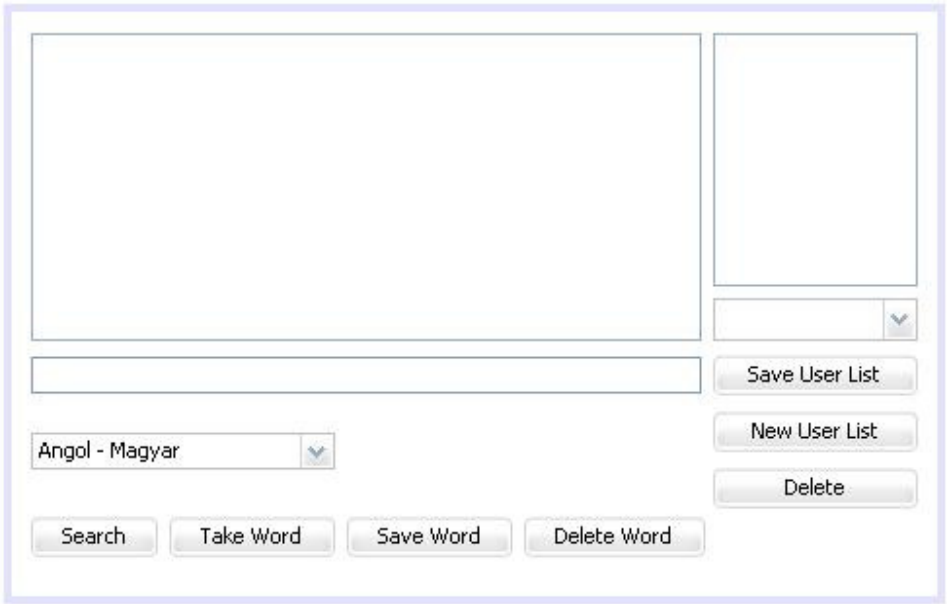
A felület tervezése

A felület két navigációs részből fog állni. A egyik a főablak tetején elhelyezkedő menüsor lesz, a másik a főbb funkciókat elkülönítő lapozó sor. A felület tervezéséhez elengedhetetlen valamilyen grafikus eszköz, mely segítségével könnyen és gyorsan alakíthatjuk, változtathatjuk elképzelésünk.

A főbb funkciókat különböző fülekre teszem majd, ezzel megkönnyítve az elérhetőségüket. Így könnyen foglalkozhatok a funkciókkal külön-külön anélkül, hogy zavar támadna az átláthatóságban.

Szótár

A szótár a program központi funkciója. A felhasználó szavakat kereshet, menthet, listákat hozhat létre rajta. Fontos, hogy megfelelően legyenek elrendezve rajta az elemek, ne nehezítsék, hanem megkönnyítsék a használatot.



1.ábra

Keresés (Search)
A szótár legfontosabb funkciója a keresés. A felhasználónak lehetősége lesz keresni egy adott szóra az adatbázisban. A keresés nem case sensitive vagyis nem különböztet meg kis és nagy betűket.

Új szó felvétele listára (Take Word)
A felhasználói listát azért alkalmazom mert így lehetőség van saját témaköröket kialakítani, mely megkönnyíti a szavak tanulását. Fejlesztői szemszögből pedig jó példa a Swing lista bemutatására.

Szó törlése (Delete word)
Szavak törlése csak felhasználói listáról lesz lehetséges, adatbázisból nem.

Szó mentése adatbázisba (Save word)
A törléssel ellentétben a felhasználó is képes lesz szavakat menteni adatbázisba.

Felhasználói lista mentése (Save user list)
A listákat adatbázisba tudjuk menteni, ezeket később vissza tudjuk tölteni, és esetleg felkérdezésekben használni.

Új Felhasználói lista létrehozása (New user list)
Mielőtt a saját szavakat listára raknánk létre kell hozni egy üres listát, melyet névvel azonosítunk.

Felhasználói lista törlése (Delete user list)
Ha már nem használunk egy listát azt törölhetjük, ilyenkor a hozzá tartozó szavak is el fognak veszni.

Kommunikációs ablak
A szótár központi eleme a kommunikációs ablak. A felhasználó itt üzeneteket kap a háttérben történő műveletekkel kapcsolatosan. Ha például egy szóra keres itt

látja az eredményt, ha töröl itt kap a törlésről megerősítést, vagy ha betölt egy listát itt megjelenik a hozzá kapcsolódó üzenet.

Felhasználói lista ablak
Ha betöltöttünk egy listát akkor a hozzá tartózó szavak itt kerülnek megjelenítésre. Ezen a listán a felhasználó ki tudja jelölni a szavak bármelyikét, és vele műveleteket végezhet.

Nyelv választó legördülő menü
Ezen legördülő-menüből választhat fordítási irányt a felhasználó.

Szó felkérdező

Ez a funkció lehetővé teszi, hogy valamelyest kipróbálja a tudását a felhasználó. Egy előre megadott listáról választ szavakat. A megjelenített szóról a felhasználónak el kell döntenie, hogy ismeri-e, és ennek tükrében megnyomni a „Tudom” vagy a „Nem tudom” gombot. A program feljegyzi a megtanult szavakat, és készít egy kisebb statisztikát.

2.ábra

Tudom gomb

A megjelenő szóra a felhasználó kétféleképpen reagálhat. Abban az esetben ha ismeri a kérdéses szót a Tudom (I know) gombot kell használni, és a program ezt könyveli is.

Nem tudom gomb

Ellenkező esetben a nem tudom gombot kell használnia (I don't know) , ezt hibának értékeli a program.

Nyelv választó

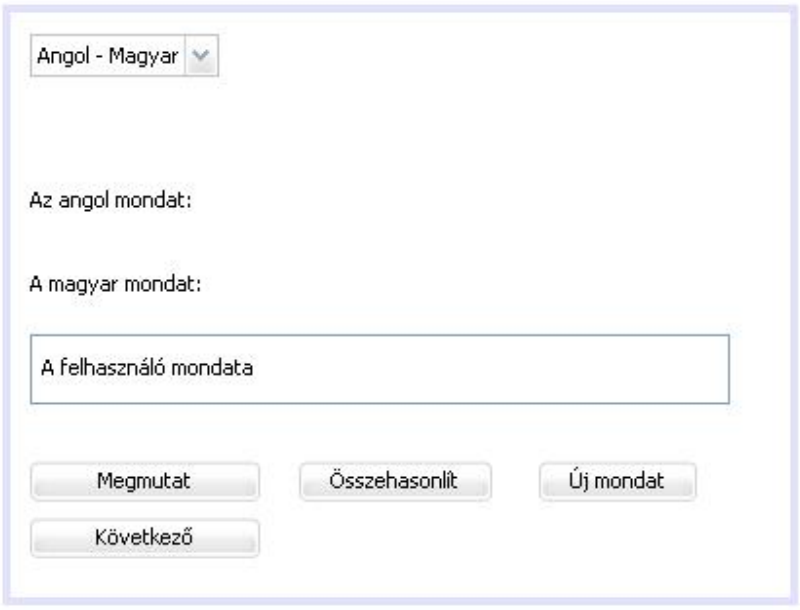
A fordítás irányát meghatározó rádiógomb.

Statisztika megjelenítő

Egy egyszerű számítás eredményét jeleníti meg, melyben a helyes és a rossz válaszok aránya szerepel.

Fordítási gyakorlat

Hasonlóan a Felkérdező funkcióhoz a Fordítási gyakorlat is a felhasználó tudását hivatott tesztelni. Itt már teljes mondatokat kell átfordítani a két nyelv között. A fordítás sikerességét a háttérben működő mintaillesztési algoritmus fogja eldönteni, melyre különös gondot kell fordítani, hogy megközelítőleg helyes eredményt adjon.



3.ábra

Nyelv választó
A fordítás irányát meghatározó rádiógomb.

Angol mondat mező
Ezután a címke után fog megjeleni az aktuális mondat angol megfelelője. Ha Angol-magyar módban vagyunk akkor már a folyamat legelején, ha Magyar-angol módban akkor csak a „Megmutat” billentyű megnyomása után.

Magyar mondat
Ezután a címke után fog megjeleni az aktuális mondat magyar megfelelője. Ha Magyar-angol módban vagyunk akkor már a folyamat legelején, ha Angol-magyar módban akkor csak a „Megmutat” billentyű megnyomása után

A felhasználó mondata mező
Ebbe az input mezőbe írhatja be a felhasználó a saját megoldását, ez a szöveg kerül majd összehasonlításra a helyes megoldással.

Összehasonlítás gomb
Hatására egy szöveg-összehasonlítás történik a háttérben a felhasználó által megadott és az eltárolt mondatra nézve. Az összehasonlítás eredményéről a felhasználót értesítjük.

Megmutat gomb
Hatására megjelenik a mondat másik nyelven való megfelelője.

Új mondat gomb
Hatására egy dialógus ablak jelenik meg, ahol új mondatot definiálhat a felhasználó. Ekkor meg kell adnia mindkét nyelven való megfelelőjét.

Következő gomb
Eme gomb hatására átugorjuk az aktuális mondatot és új mondat jelenik meg feladatként.

Menüből elérhető funkciók

Néhány hagyományos funkción kívül (exit, copy, paste) csak egy-két funkció kerül ide. Ilyen például, hogy a program felületének nyelvezetét tudjuk vezérelni. A beállítások \ nyelv menü segítségével válthatunk angol és magyar között.

Implementáció

Implementáció során az általam helyesnek tartott sorrend szerint először a program működéséhez szükséges adatstruktúrát kell megalkotni – ezzel párhuzamosan akár az adatbázisban felépíteni a táblákat -, ezt követően már foglalkozhatunk a funkcionalitással. Minden komolyabb adattípushoz létrehozok egy Manager osztályt, amely majd ellátja a vele kapcsolatos teendőket (beolvasás, törlés, keresés, stb.). Mivel ezek az osztályok valamelyest rokon tulajdonságokkal bírnak – abból adódóan, hogy mindannyian adatbázis elérést is megvalósítanak - célszerű megalkotni először egy olyan osztályt mely elvégzi ezt a tevékenységet és a többi osztályt ennek leszármazottjaként hozzuk majd létre.

Adatstruktúra

A *Model* csomagba rendeztem azokat az osztályokat amelyek a program adatstruktúrájáért felelősek.

Database osztály

Ez az osztály fogja segíteni az adatbázissal való kapcsolatot. Mindig célszerű egy ilyen osztály használata mely példányosításkor megkapja az adatbázis eléréshez szükséges információkat (URL, password, user) és aztán tőle egyszerűen kérhetünk kapcsolat objektumot.

Lista osztály

Egy egyszerű lista osztály annyi különbséggel, hogy példányai már tudni fogják magukról, hogy mely felhasználóhoz tartoznak.

Sentence osztály

A egy ilyen osztály példányai fogják megtestesíteni felkérdezések során használt mondatainkat; tárolják mind a magyar mind az angol változatot.

User osztály
Az egyes felhasználók absztrakciója lesz ez az osztály. Leghasznosabb tulajdonsága, hogy tárolja majd a felhasználói nevet, de ezen kívül képes egy e-mail cím, egy jelszó megőrzésére. Bár több-felhasználós programot készítettem, nem lesz komoly szerepe az authentikációnak.

Word osztály
A program adatszerkezetének alapkövét adó osztály a feldolgozandó szavak angol és magyar nyelvű változatát tárolja majd.

Az osztályok biztonságos használata érdekében mindenütt private tagok szerepelnek, és ezeket csak a szükséges mérték érik el a rajtuk kívüli funkciók.

Feldolgozás
A *Proccessing* csomagba kerültek azok az osztályok amelyek a program funkcionalitásáért felelősek.

Manager osztály
Ahogy már említettem létrehoztam egy ősosztályt, amely definiál néhány metódust melyek közül a következők a fontosabbak:

- rollback : erre akkor lesz szükség ha feldolgozás során valamilyen problémába ütközünk, ekkor kivétel dobódik és eme metódus segítségével visszagörgethetjük az aktuális

  tranzakciót, semmisé téve a változtatásokat.

- close : segítségével lezárhatjuk a már nem használt adatbázis kapcsolatokat.

ListManager osztály

Ez osztály a felhasználói listával kapcsolatos adatbázis-műveletekért lesz felelős. A következő műveleteket valósítja meg az osztály:

Save

A Lista osztály egy példányát kapja paraméterül, és azt adatbázisba menti. A háttérben az SQL INSERT utasítás dolgozik. Miután elkértük Database osztály egy példányától a kapcsolatot azon létrehozunk egy PreparedStatement objektumot.

Ezt megfelelő paraméterezés után végrehajtva bekerülnek az adatbázisba a kívánt sorok. Kivétel dobása esetén azt elkapjuk és visszagörgetjük a tranzakciót. Ha nem történt hiba akkor Commit –tálunk és lezárjuk a kapcsolatot.

Create

Az itt következő Create metódusra azért van szükség, mert mikor a listát adatbázisból beolvassuk (Load) akkor az még csak szavak egy halmaza, ez az függvény pedig elkészíti belőlük a tényleges Lista objektumot.

Load

A Lista objektumok azonosítóval, névvel rendelkeznek. A Load metódus ezt a nevet paraméterül kapja és egy SQL SELECT segítségével kiválogatja a listához tartozó szavakat. Egy táblát ad vissza amely egy ResultSet objektumba kerül. Ez az objetum lesz paramétere az előbb már említett Create metódusnak mely Listát készít belőle.

Exists

Ezzel a metódussal azt ellenőrizhetjük, hogy létezik-e már egy szó az adott listában, a duplikációk elkerülése végett. Egy lekérdezés a metódus kulcsa mely

megkapja a tesztelendő szót és kiszelektálja a listából. A háttérben itt is SQL SELECT metódust alkalmazok.

Delete
Egy szót tudunk kitörölni az adott felhasználói listából. A PrepareStatement-ben felparaméterezett

SentenceManager osztály
Ez az osztály a mondatokkal kapcsolatos adatbázis és egyéb műveletekért felelős.

Save
Hasonlóan működik mint a ListManager osztály Save metódusa. A különbség csak annyi, hogy nem Listákat hanem Sentecence objektumokat mentünk az adatbázisba.

Load
Hasonlóan működik mint a ListManager osztály Load metódusa. Ez paraméterként egy egész számot kap, mely majd meghatározza hogy hány mondatot szeretnénk betölteni.

Delete
A Sentence objektumok azonosítóval (id) rendelkeznek. Ezzel tudjuk őket megfogni, és törölni az adatbázisból.

Search
Kétféle kereső eljárást valósítottam meg, melyek inverzei egymásnak oly módon, hogy az egyik paramétere az angol mondta, eredménye a magyar, és fordítva.

UserManager osztály
A UserManager osztály a felhasználók karbantartásáért felelős.

Save
Egy felhasználót ment az adatbázisba. Működése hasonlatos az ListManager osztályban leírtakéhoz.

Create
Egy felhasználót hoz létre az adatbázisból kinyert adatokból és ezt egy User objektum formájában a hivó felé továbbítja.

Getuserid
A felhasználóhoz tartozó azonosítót kéri el az adtabázisból. Paraméterként a felhasználó nevét kell megadni.

List
Az adatbázisban tárolt felhasználókból készít egy listát. Felhasználja a Create metódust így a visszaadott lista már User objektumok listája lesz.

WordManager osztály
Végül de egyátalán nem utolsósorban a WordManager osztály. Jelentős szerepe van hiszen a központi adattípuson fog műveleteket végezni, ezért érdemes jól optimalizáltra írni a metódusait, mivel arra számítunk, hogy gyakran fognak meghívódni. Főbb funkciói a következők:

Save

Egy szó objektumot ment adatbázisba. Ne feledjük, hogy itt a szó objektum az adott szó minden nyelven való megfelelőjét jelenti. Hasonlóan - ahogy a többi Manager osztály is – itt is az SQL Insert utasítás dolgozik a háttérben.

Search

Itt egy helyen említeném a Search alatt a mindkét irányú keresést. Én az utat választottam, hogy mindkét irányban definiáltam egy kereső függvényt. Igaz, hogy ezzel látszólag némi többlet munkát végeztem, de ez megtérülhet ha a kereséseket specializálni akarjuk valamelyik irányban.

Load

Ez a függvény nem kifejezetten egy szó betöltésére való (hiszen ezt elvégzi valójában a Search is) hanem szavak egy halmazának a betöltésére. Erre akkor lehet szükség ha például egy listát szeretnék összeállítani, de hasznosnak bizonyulhat más esetekben is.

Delete

Szavaink az adatbázisban rendelkeznek egy azonosítóval. Ez mindegyik szó esetén különböző, ha úgy tetszik elsődleges kulcs. A Delete ennél fogva tudja törölni az adatbázisból az adott szót.

Megjelenítés

A *Frames* csomagba kerülnek az osztályok amelyek a megjelenítésért és az interakcióért felelősek.

Course osztály

Ez az osztály felelős a szó felkérdező megjelenítéséért. Ezenkívül még tartalmaz hivatkozást az aktuális szóra, az aktuális listára, számolja és nyilvántartja a felhasználó eredményeit. Azonkívül, hogy kirajzolja a különböző gombokat, szövegmezőket és címkéket az ő feladata, hogy megvalósítsa az interaktivitást. Ezt ActionListener –eken keresztül teszi. Az ilyen Listenerek definiálnak egy reakciót, amely adott esemény bekövetkeztekor (gomb lenyomása) végrehajtódik.

Ebben az osztályban ilyen például a „Megmutat” gomb hatása, mikor is megjelenítésre kerül az adott mondat a kívánt nyelven.

Dictionary osztály

Ez az osztály felelős a szótár megjelenítéséért. Ezentúl még tartalmaz hivatkozást az aktuális szóra, az aktuális listára, és a felhasználóra, felhasználói azonosító formában. A megjelenítésen kívül az ő feladata, hogy megvalósítsa az interaktivitást. Ezt ActionListener –eken keresztül teszi. Az ilyen Listenerek definiálnak egy reakciót, amely adott esemény bekövetkeztekor (gomb lenyomása) végrehajtódik.

Ebben az osztályban ilyen például a „Keresés” gomb hatása, mikor is egy keresés megy végbe a már előre megadott szóra.

MainFrame osztály

Ez azaz osztály ahol a main metódusunk is helyet foglal. Ő sorrendben a következőket csinálja: elindítja az adatbázis szervert, példányosít egy Jframe-t, mely a megjelenítés alapja lesz, és beállítja az alapértelmezés szerinti nyelvet. Ezentúl még néhány apróbb paramétert is beállít, mint az ablak méret és láthatósága.

MenuRow osztály

A menüsor reprezentálására szolgáló osztály. Hierarchiában tartalmazza a menü-itemeket, és a hozzájuk kapcsolódó műveleteket.

newDialog osztály

Ez az osztály valamelyest univerzális. Arra használom, hogy ha a felhasználóval szeretnék valamilyen dialógust folytatni. Ilyen helyzetek például mikor új felhasználót, vagy új listát, vagy új mondatot szeretnék létrehozni.

Sentences osztály
A fordítási gyakorlat megjelenítéséért felelős osztály. A Course és a Dictionary osztályhoz hasonlóan tartalmazza a szükséges felületelemeket és a mögöttük rejtőző interaktivitásért felelős kódot.

Util csomag

Egy Util csomag is létrejött mely azokat az osztályokat tartalmazza amelyek nem kötődnek szorosan egyik eddig említett funkciócsoporthoz sem.

Language osztály
Ezen osztály segítségével valósítom meg a felület többnyelvűségét. Egy fájlból olvassa be az egyes felületelemekhez rendelt neveket. Egyszerűen bővíthető - akár a létező kettőnél - több nyelvűre is.

Tesztelés

Tesztelés során a hagyományos teszteseteken kívül érdemes olyan esetekre is koncentrálni amelyek szokatlanok és csak extrém esetekben fordulnak elő. A kellő odafigyeléssel kell kezelni a felhasználói inputot, érdemes mindig tesztelni a bejövő információt mielőtt odaadnánk a háttérben működő logikának.

Például tegyük fel, hogy egy szóra szeretnénk keresni az adatbázisban. A keresendő szót (mint sztringet) továbbadjuk a kereső eljárásnak, ami paraméterül adja egy olyan PrepareStatement –nek amiben egy SQL Select szerepel. Ez az eset olyan mintha közvetlenül beleírnánk a szót a Select –be. Na már most ha nem figyeltünk kellően a Select megírásánál, és a felhasználó jól ismeri az SQL nyelvet akkor könnyedén félrevezetheti a keresőeljárásunk. Gondoljunk csak arra, hogy a lekérdezésben ’like’-ot használtunk, és felhasználó ismeri az SQL

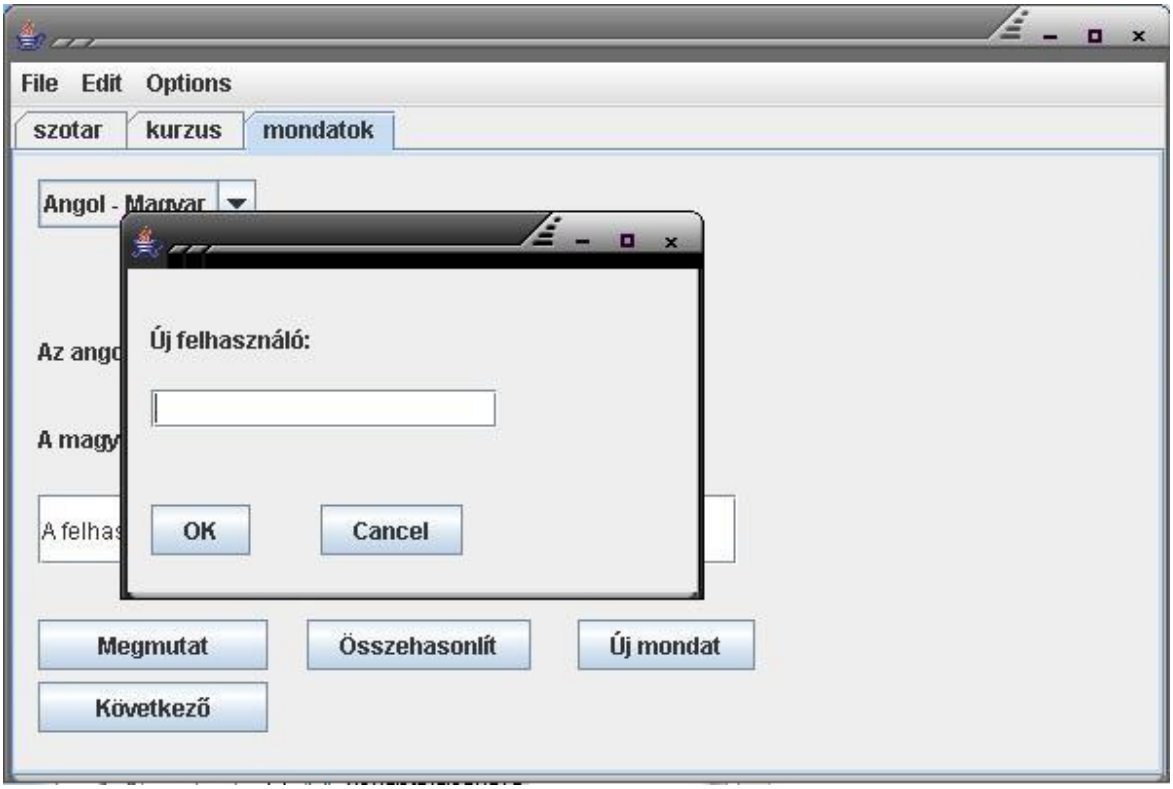
joker karaktereit. Ez a példa talán túlzó egy ilyen program esetében, de azt hiszem érdekes, és mindenképpen rávilágít az esetleges téves működésekre.

Itt nekünk elsősorban arra kell ügyelni, hogy ha a felhasználó számot ad meg akkor mi számként kezeljük, ha szöveget akkor szövegként. Érdemes a felhasználó lehetőségeit olyan mértékűre csökkenteni amelyen még kényelmesen tudja használni a programot , de kis lehetősége marad hibázni. Ez alatt azt értem, hogy ha valamilyen objektumot be szeretne tölteni akkor ne kézzel kelljen megadnia a objektum nevét, hanem mondjuk egy legördülő lista segítségével. Ez kisebb lista esetén nem jelentene gondot, de például ha egy város listáról van szó mindenképp érdemes.

A funkciók végignyomogatással, és próbálgatással viszonylag egyszerűen tesztelhetőek. Ami komolyabb feladatot jelenthet azaz úgynevezett stressz-teszt. Itt már arra kellene felkészíteni a programot akár több 1000 vagy 10000 szó kezelésére. Ez a program azonban csak demonstrációs célokat szolgál így ettől eltekintettem.

Felhasználói kézikönyv

Első indításkor meg kell adnunk egy felhasználói nevet, mellyel a program ezután azonosítani fog minket. Az ezt követő indulásokkor már csak egy listából kell kiválasztanunk, hogy melyik felhasználóval szeretnénk használni a programot.



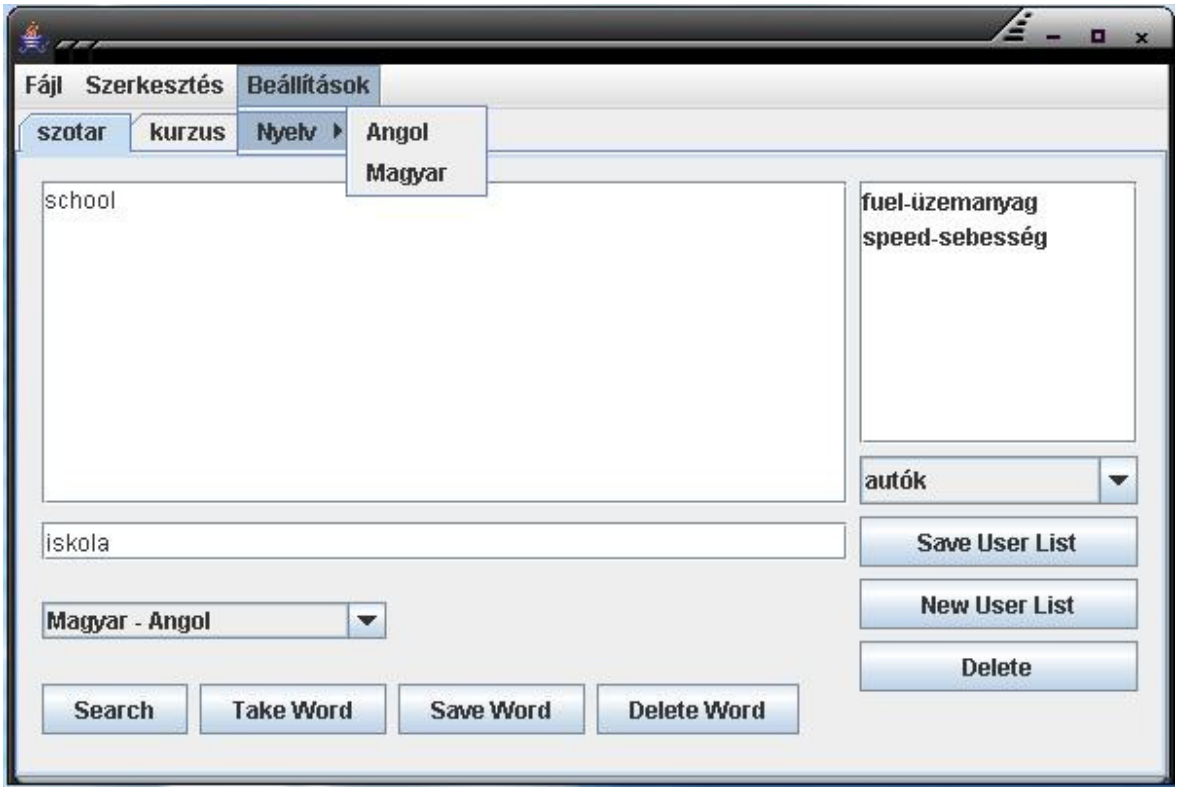
4.ábra

A szótár funkció használata

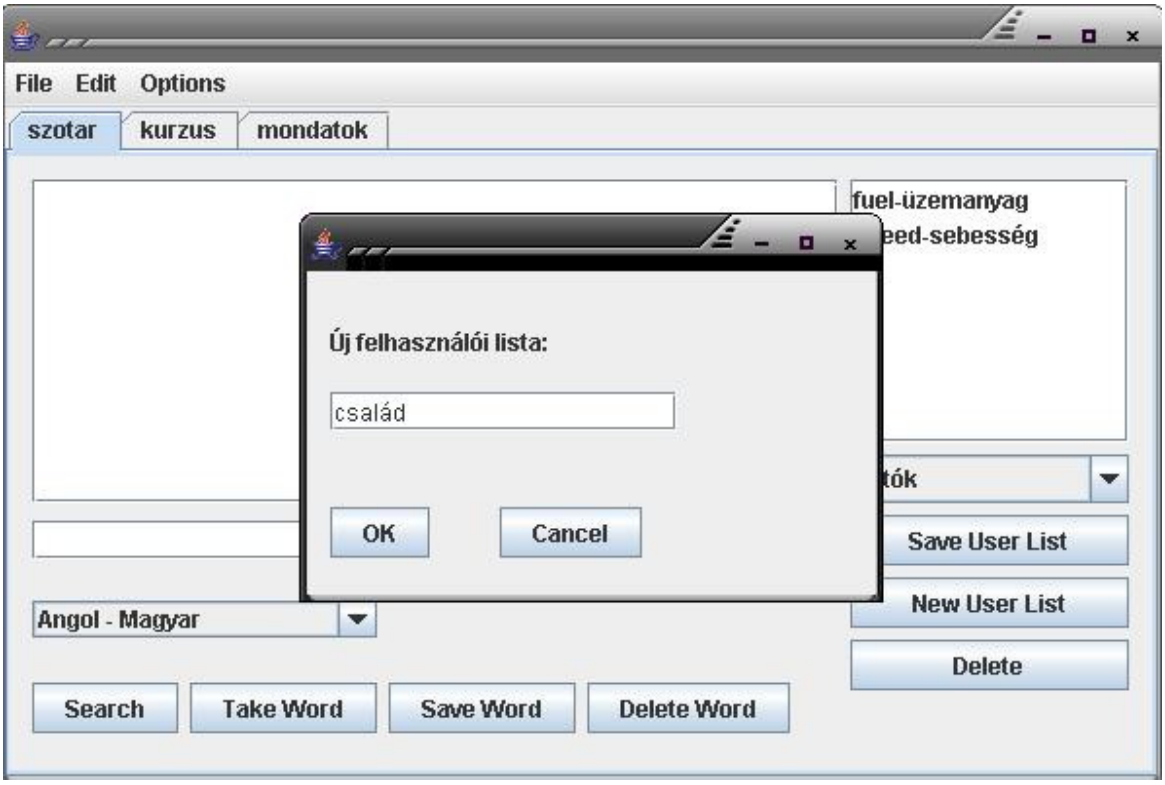
Egy szó kereséséhez előbb be kell állítani a fordítási irányt, majd beírni a szót a szövegmezőbe, majd megnyomni a Keresés (Search) gombot. Ekkor kétféle üzenet jelenhet meg a képernyőn. Egyik esetben – ha megtalálta – megjeleni a szó a másik nyelven. Ellenkező esetben hibaüzenet kapunk arról, hogy a szó nem található az adatbázisban. Ha megtalálta a szót akkor lehetőség van ezt egy

felhasználói listára küldeni a Take Word gombbal. Figyeljünk arra, hogy ilyenkor az aktuálisan megnyitott listára kerül a szó. Továbbá lehetőség van törölni az aktuális szót a Delete Word gombbal.

Ha még esetleg nem létezne lista akkor a New User List billentyűvel készíthetünk egyet. Ilyenkor csak a lista nevét kell megadni és kapunk egy üres listát. Ezt bármikor törölhetjük a Delete gombbal. Figyeljünk arra, hogy ezek a listák csak akkor mentődnek adatbázisba ha használjuk a Save User List billentyűt.



5.ábra

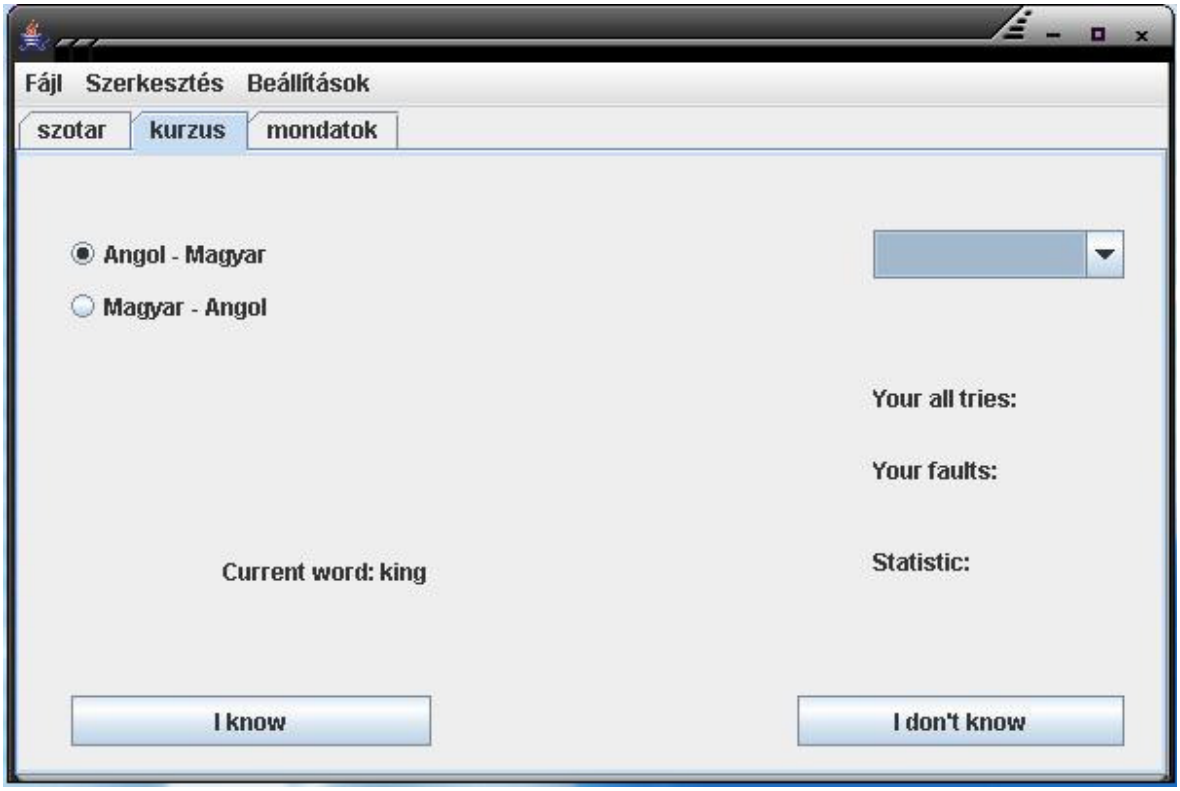


6.ábra

A szó felkérdező vagy kurzus funkció használata

A szavak felkérdezésénél először beállítjuk a fordítási irányt. Ekkor már előttünk is van az első kérdéses szó. Ha ismerjük a szót akkor a Tudom gombot kell nyomni, ellenkező esetben a Nem Tudom gombot. Ha azt szeretnék, hogy csak egy bizonyos témakörön belül kapjunk szavakat akkor ki kell választanunk egy listát. A program folyamatosan figyeli teljesítményünk és arról statisztikát

készít.



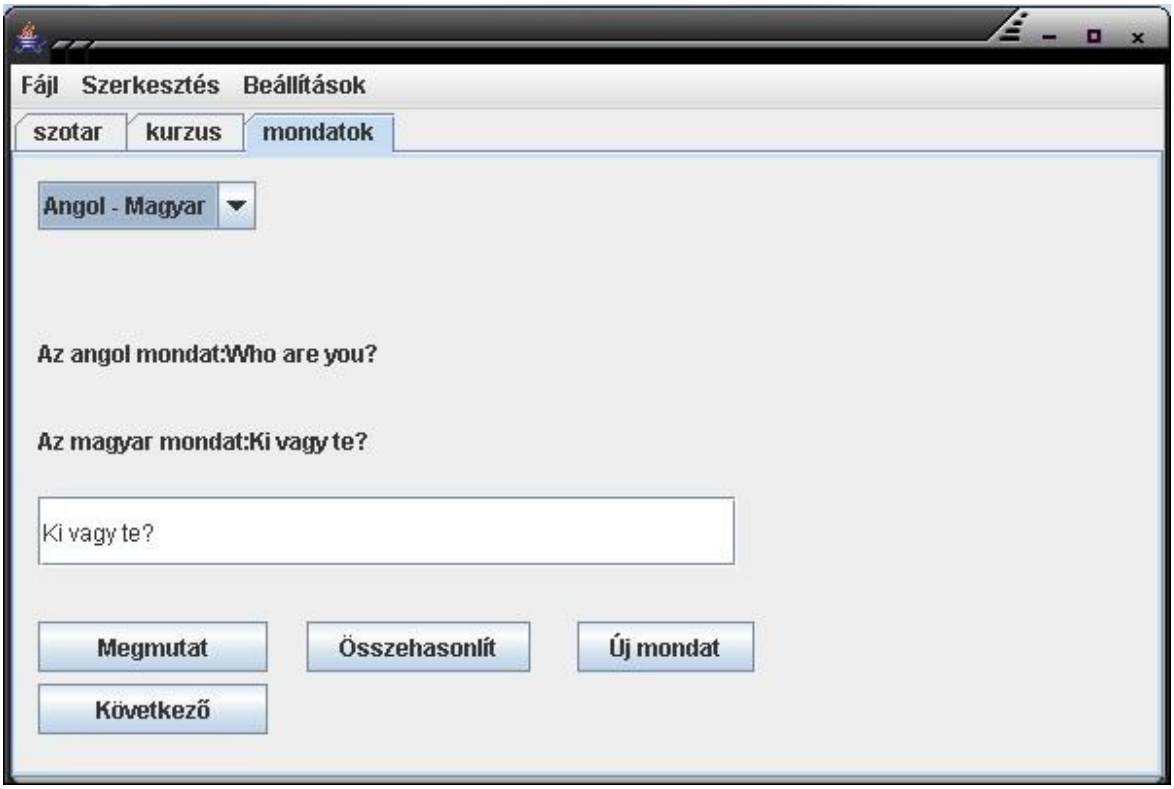
7.ábra

A fordítási gyakorlat használata

A használat szintén a fordítási irány kiválasztásával történik. Ezután kapjuk a fordítandó mondatot, és a szövegmezőbe beírhatjuk megoldásunkat. Miután ezt megtettük a Megmutat gombbal megnézhetjük a helyes megoldást. Lehetőség van a két változat összehasonlítására, így megláthatjuk a kisebb eltéréseket.

Ha szeretnénk új mondatot bevinni az adatbázisba azt az Új mondat gombbal tehetjük meg. A gyakorlat folyatatásához a Következő gombot kell

megnyomni.



8.ábra

Összefoglaló

Úgy gondolom elértem célom, hogy bemutassam eme program kifejlesztését az adott környezetben, és talán kedvet csináljak hasonló tervek megvalósításához. Reményeim szerint hasznos ismeretekhez jutattam az olvasót, és tudja majd ezeket kamatoztatni.

A program elkészült és átadtható a felhasználóknak. Bárkinek aki idegennyelvvel ismerkedik vagy a szókincsét szeretnék gyarapítani hasznos segítséget nyújtahat eme program. A tanulás hatásfokát jelentősen fokozhatja a megismert szavak visszakérdezése, a programban erre is van lehetőség. A használathoz kívánok kellemes perceket.

A magam részéről köszönetet kell mondanom témavezetőmnek, a hozzám való türelméért, és értékes útmutatásaiért. Az olvasónak pedig ezúton kívánok sok sikert eme témakörhöz.

Irodalomjegyzék

- Java *2* útikalauz programozóknak I.-II.
- Sun Java 1.4 documentation. www.java.sun.com/j2se/1.4/docs
- Java 2 Platform, www.java.sun.com
- The Java Tutorial, http://java.sun.com/docs/books/tutorial/
- SQL–99 complete, Really, http://www.ocelot.ca/sql99.htm
- Hsqldb http://hsqldb.org/