SZAKDOLGOZAT

Szauervein Szabolcs

Debrecen

2009

Debreceni Egyetem

Informatika Kar

Keretrendszer fejlesztés Java nyelven

Témavezetõ:

Espák Miklós

egyetemi tanársegéd

Készítette:

Szauervein Szabolcs

Programtervezõ informatikus (BSc)

Debrecen

2009

Keretrendszer fejlesztés Java nyelven

A HiberGUI keretrendszer

Tartalomjegyzék

1 Bevezetés

A modern szoftverfejlesztési eszközöknél fontos szempont az adatbázis, a hatékony fejlesztés és az igényes felhasználói felület támogatása. Ennek is köszönhetõ a Java Platform máig tartó töretlen sikere. A szakdolgozatom tartalmát ebben a témakörben helyeztem el. Célja egy, az adatbázist használó, asztali alkalmazások fejlesztését megkönnyítõ, újrafelhasználhatóságot, könnyû és gyors fejlesztést lehetõvé tevõ keretrendszer, a HiberGUI fejlesztése.

A szakdolgozat elsõ fejezetében ízelítõt kapunk a használt Java technológiákról, ami után a rendszer szerkezetének és mûködésének ismertetése történik. A következõ fejezetben a fejlesztést megelõzõ szükséges kódtisztítások és átszervezések következnek. Végül az egyedek közti kapcsolatok általános tárgyalása, és a keretrendszer kapcsolatok kezelésénél felmerülõ problémájának és annak megoldásának ismertetése következik.

2 Használt eszközök és technológiák

2.1 Java EE

A Java Platform, Enterprise Edition (Java EE) válogatott Java technológiák halmaza, melyek jelentõsen csökkentik a költségeit és komplexitását a többrétegû, szerver központú alkalmazások fejlesztésének és menedzselésének. A Java EE a Java Platform, Standard Edition-ra (Java SE) épülve, azt kiegészítve teljes, stabil, biztonságos és gyors Java Platformot nyújt a vállalati igények kiszolgálására. A jelenlegi Java EE technológiák a következõ csoportokba sorolhatók: Web Services Technologies , Web Application Technologies, Enterprise Application Technologies, és Management and Security Technologies.[1]

Ezek közül számunkra a Java Persistence API lesz fontos.

2.2 Java Persistence API

A Java Persistance API (JPA) egy POJO alapú perzisztencia kezelési modell, mellyel megvalósítható az objektum-relációs leképezés. A POJO(Plain Old Java Object) Közönséges Java objektumot jelent. Gyûjtõnév, mely alá olyan objektumok tartoznak, melyeknek nem szükséges megfelelniük egy keretrendszer elõírásainak. Ellentétben például az Enterprise Java Bean-ekkel, melyek erõsen kötõdnek az Enterprise Java Bean technológiához.[2]

A JPA-t az EJB 3.0 szoftver szakértõi csoport fejlesztette ki a JSR 220 Java Specification Request részként, de a használata nincs az EJB szoftver komponensekre korlátozva. Használható web alkalmazások vagy alkalmazás kliensek által, de a Java EE platformon kívül is, például egy Java SE alkalmazásban.

A JPA-nak számos implementációja létezik például Hibernate és a TopLink.

2.3 Hibernate

A Hibernate egy nagy teljesítményû objektum/relációs perzisztencia és lekérdezõ szolgáltatás. Lehetõvé teszi, hogy a fejlesztés során az objektum-orientált idiómák követését - többek között az asszociációt, öröklõdést, polimorfizmust, kompozíciót és kollekciókat. A lekérdezések pedig kifejezhetõk a saját hordozható nyelvû SQL kiterjesztéssel, szintúgy natív SQL-el vagy az objektum-orientált Criteria API segítségével.[3]

2.4 Eclipse Platform

Egy komponens alapú integrált fejlesztõi környezet. Eredetileg Java-ban történõ fejlesztésre lett létrehozva, de az óta számos egyéb nyelvet támogat. Jelenleg az egyik legmodernebb IDE. A kód kiegészítéstõl, a gépelés közbeni automatikus fordításon át a végletekig személyre szabható felületig, az XML dokumentumok írásától az Google Android mobiltelefonos operációs rendszer projektekig számos funkcióval bír. Mivel nyílt forráskódú, ingyenes és központi szereppel bír benne a bõvíthetõség, ezért nem meglepõ, hogy a nyílt forráskódú fejlesztõeszközök népszerû platformja.

2.5 Standard Widget Toolkit

A Standard Widget Toolkit (SWT) egy nyílt forráskódú eszközrendszer, ami hatékony és hordozható hozzáférést biztosít az õt implementáló operációs rendszer felhasználói-felületéhez. Korábban az IBM égisze alatt fejlesztették, most az Eclipse Foundation fejleszti.

2.6 Apache Ant

Programozás közben felmerülõ feladatok automatizálását megkönnyítõ eszköz. A Linux operációsrendszereknél ismert GNU make helyét tölti be a Java világban. XML alapú konfigurációs állományokat használ és platform független.

2.7 HSQLDB

A HSQLDB (HyperSQL DataBase)[1] egy adatbázis kezelõ rendszer, mely vezetõ szerepet tölt be a Java nyelven írt SQL relációs adatbázisok közt. Rendelkezik JDBC meghajtó programmal, és az ANSI-92 SQL jelentõs részhalmazát támogatja. Kicsi (az appletekhez szánt verzió kevesebb, mint 100KB) és gyors adatbázis motor, ami rendelkezik mind memóriabeli, mind háttértárbeli táblákkal. Támogatja a beágyazott és a szerverként történõ telepítést is.

Jelenleg számos nyílt forráskódú vagy kereskedelmi szoftver adatbázisaként és perzisztencia kezelõ motorjaként használják. A legújabb kiadás pedig különösen stabil és magas rendelkezésre állással bír. Legtöbben a kis méretérõl, teljes memóriabeli futási képességérõl, rugalmasságáról és sebességérõl ismerik.

Jó megoldás adatbázist használó programok teszteléshez is.

---

1 http://hsqldb.org/

3 A meglévõ rendszer ismertetése

Egy meglévõ rendszer fejlesztésének elsõ lépéseként azt meg kell ismerni. A cél a HiberGUI mûködésének és használatának ismertetése. Ez magában foglalja a késõbbi munka elõkészítése céljából történõ plusz tervek elkészítését is.

3.1 Bevezetés HiberGUI-ba

A HiberGUI középpontjában a Java entitások állnak. A feladatuk az adatmodell megvalósítása a Java program szintjén. Ezen objektumok jellemzõit konvencionális módon a lekérdezõ és beállító metódusok segítségével lehet elérni. A többi entitással való viszonyukat pedig a kapcsolódó objektum referenciájával, vagy több objektum esetén azokat tartalmazó kollekcióval lehet leírni.

Ha ezek az objektumok eleget tesznek a JavaBeans Specifikációjának, akkor bean-eknek nevezzük õket. A bean-ek fontosabb jellemzõik, hogy rendelkeznek publikus, üres paraméter listájú konstruktorral, és az attribútumait a specifikációban meghatározott deklarációjú metódusokkal kérdezzük le és állítjuk be. Az elsõ kikötés biztosítja az bean-ek reflexió segítségével történõ példányosításának egyszerûségét és azt, hogy a példányosításkor nem szükséges az objektum mezõit beállítani. Ezeket lehet késõbb a megfelelõ helyen és idõben megtenni. Erre szolgálnak a beállító metódusok. Ha ezeket konvenció szerint készítjük el, akkor reflexív módszerekkel egyszerûen tudjuk módosítani az objektum állapotát. Az egyes attribútumok értékeinek lekérdezését pedig a lekérdezõ metódusok hívásával tehetjük meg.

Az adataink perzisztens tárolásához adatbázisra van szükségünk. A bean-ekben tárolt adatokat a perzisztenssé tétel érdekében az adatbázisban lévõ relációs adatmodellünkben is le kell tárolni. Itt kerül szembe a Java objektum orientált adatmodelljének és az adatbázis relációs adatmodelljének reprezentációs különbsége. Az objektum orientált adatmodellben az egyedek objektumokként, a relációs modellben rekordokként vannak tárolva. Az elsõben az egyedtípust az osztály, a másodikban a reláció jelenti. Az elsõben a kapcsolatokat objektum referenciákkal, míg a másikban hivatkozási integritási megszorításokkal érjük el.

Az objektum-orientált és a relációs modell közti különbség leküzdésére fejlesztették ki az objektum/relációs eszközöket. Ilyen keretrendszer a Hibernate is. Segítségével a nem kell a fejlesztõknek a leképezés megvalósításával foglalkoznia. Elég bekonfigurálni, a leképezési adatokat megadni és az API által biztosított felületrõl kezelni az adatbázisban lévõ adatokat.

A HiberGUI ezt a gondolatmenetet viszi tovább a grafikus komponensek felé. Az alkalmazások grafikus felülete nagyban tükrözi az alattuk megbúvó entitások szerkezetét.

A grafikus felület egyes adatokat megjelenítõ vagy adatbevitellel kapcsolatos komponenseihez a háttérben általában egy entitás attribútumai tartoznak hozzá. De megvan a kapcsolatoknak, és az alkalmazás logikai mûveleteknek is a felületen való megjelenése is. Természetesen itt is, mint az objektum/relációs leképezésnél, szükség van a két adatreprezentáció közti átalakításra. A felületen megjelenõ adatokat át kell alakítani a JavaBean-ek szintjén megjelenõ adatokra és vissza. Ennek a leképezésnek a megkönnyítése a célja HiberGUI keretrendszernek. Az adatok mellett mûveletek is találhatók a grafikus felületen melyeket a beaneken végzett mûveletekre kell leképezni.

3.2 A HiberGUI szerkezete

A HiberGUI MVC tervezési mintát követi. A minta szerint a HiberGUI több, egymástól jól elkülöníthetõ részre bontható.

Modell

- *Az adatmodel*: JavaBean-ek alkotják. Kiegészítve leképezési adatokkal, ami vagy XML-el vagy annotációkkal van megadva. Ezt teljes egészében a keretrendszer használójának kell elkészítenie.
- *Az üzleti logika mûveletei*: Az entitásokon végeznek mûveleteket. Az Létrehozás, Lekérdezés, Módosítás, Törlés mûveletekhez a keretrendszer nyújt kész megoldásokat. Ezek megvalósítása a Hibernate-on keresztül történik. Az entitásokon végzett mûveletek a Hibernate közvetítésével az adatbázisban is végrehajtódnak. A bonyolultabb mûveleteket a felhasználónak kell megvalósítania.

Megjelenítés

SWT komponensek által történik. Minden HiberGUI komponens a Component osztály leszármazottja. Az teljes entitásokat kezelõ komponensek a Form, míg attribútumokat és kapcsolatokat kezelõk az Edit osztály leszármazottjai. Ahogy az objektum részét képezik az attribútumok, úgy a Form-oknak részét képezik az Edit-ek a felhasználói felületen. Az Editeken kívül a Form-okban lehetnek a keretrendszerbe beépített eseményeket kiváltó komponensek, például gombok. A tranziens perzisztencia kezelés miatt az entitásokat egyszerû JavaBean-ként kezelhetjük.

Vezérlés

Esemény alapú. Események és eseményfigyelõk kapcsolatrendszere alkotja. A mûködésben részt vesznek az SWT és a HiberGUI saját eseményei és esemény figyelõi.

Egyéb elemek

Olyan segéd objektumok és osztályok melyek nem tartoznak szorosan egyik kategóriába sem.

3.3 A HiberGUI fontosabb osztályai

A grafikus komponensek õsosztályai a *hibergui* csomagban találhatók. Ezek a következõk:

Component

Minden grafikus komponenst ebbõl kell származtatni. Rendelkezik egy T típusparaméterrel. Ez határozza meg a komponens által befogadott entitást, attribútumot, vagy egy asszociációt reprezentáló kollekció típusát. Bár grafikus elem, nem SWT widget. Nincs származtatva az SWT `Composite` osztályból. Ehelyett egy `Control` példányt tárol mely egyes metódusait delegálja. Ez a `Control` fogja meghatározni a `Composite` kinézetét.

A `Control` példány meghatározása a konstruktorban történik. A konstruktor paraméterül vár egy SWT `Composite` objektumot, amire meghívja a Control createContents(Composite) metódust. Ez pedig visszatér a beágyazandó Control objektummal. A leszármazottak feladata a createContents metódus implementálása. Ezzel a módszerrel a `Composite` már konstruálás után rendelkezni fog a megfelelõ SWT widgettel.

A HiberGUI-nak saját eseménykezelõ rendszere van. Az események, az esemény figyelõk és azok adapterei a *hibergui.event* csomagban találhatók. A befogadott `Control` eseménykezelését a `Component`-en keresztül éri el, azáltal, hogy delegálja bizonyos metódusait.

Form

Az Edit mellett a HiberGUI egyik legfontosabb osztálya. Õsosztálya a `Component`. Minden Form egy entitás kezeléséért felelõs. Ennek típusát a T típusparaméter határozza meg. A Form lehetõséget biztosít az entitás létrehozására, törlésére, módosítására. Az entitás jellemzõit és kapcsolatait az Edit objektumokon keresztül tudjuk kezelni. Egy Form több Edit példányt is tartalmazhat. Általában egy entitás minden attribútumához egy attribútum beállító Edit és minden kapcsolatához egy kapcsolat beállító Edit tartozik. Az Editeknek a grafikus felületen, mint szövegmezõk, legördülõ menük stb. jelennek meg, míg a Form egy ezeket, az

Editeket, plusz a mûveletek gombjait tartalmazó komponensként szerepel a képernyõn. Az Edit-ek egy kollekcióban vannak letárolva a Form-on belül. Az entitás módosítását, létrehozását, törlését a Form hoz tartozó megfelelõ SWT gombokkal lehet kérni.

Az Edit-ekben tárolt információkat az entitásba az extract(T) metódussal tudjuk átvinni. Ennek hatására az Edit-ekben beállított értékekre lesz beállítva a hozzájuk tartozó entiásbeli attribútum. Kapcsolatok esetén pedig a kapcsolatot reprezentáló kollekció lesz beállítva az Edit értékébõl. Az entitás példány megadására pedig a megfelelõ get() és set(T) metódus szolgál.

A Form-ban meg van határozva néhány egyszerû, de alapvetõ alkalmazás logikai mûveletek. Ezek az entiások perzisztencia kezelését végzik. Absztrakt metódus révén a gyermekek feladata a konkrét perzisztenica kezelõ implementáció megadása.

1. táblázat

Metódus szignatúrája	**Tevékenység**
commit()	Véglegesíti az entitás korábbi változatásait.
rollback()	Visszagörgeti az entitás korábbi változtatásait.
save(T)	Az entitást elmenti az adatbázisba.
update(T)	Az entitást módosítja az adatbázisban.
delete(T)	Az entitást törli az adatbázisból.

A grafikus felületen kiváltott eseményeket és e mûveleteket a Form eseménykezelõi fogják összekapcsolni.

Edit

Míg a Form segítségével egyedeket tudunk szerkeszteni, addig az Editekkel az egyedek jellemzõit, vagy azok kapcsolatait tudjuk kezelni. Az entitás durvább finomságú egység az attribútumhoz képest. Ez tükrözõdik a Form és az Edit-ek viszonyában is. Egy Form több Editet is tartalmazhat. És ez a tartalmazás a grafikus felületen is megjelenik.

Aszerint hogy attribútumot vagy kapcsolatot állít be, az Edit-eknek két fõ típusát különböztetjük meg:

- attribútum Edit: Az entitás egy attribútumának értékét adhatjuk meg vele. Az osztályai hibergui.propety csomagban találhatók. A T típusparaméter megadja, hogy milyen típusú attribútumot szerkeszthetünk vele. A levél osztályokban konkrét típust vesz fel. Például a szöveges tartalmat beállító StringEdit típusparamétere String. A dátum beállítására szolgáló DateEdit típusparamétere pedig Date. Mind a StringEdit, mind a DateEdit a grafikus felületen szövegmezõként jelenik meg. A szülõosztályok eseménykezelõ rendszerének használatával folyamatosan figyelik, hogy a bevitt adat érvényes e. Ha nem, akkor ezt jelzik a felhasználó felé.
- asszociáció Edit: Az entitás egy kapcsolatát reprezentáló entitás vagy entitás kollekció adható meg vele. A hibergui.assoc csomagban találhatók az osztályai. Közös õsosztályuk az AssociationEdit.

Az attribútumokat beállító Editek nem érik el közvetlenül az entitásokat, az Edit-ek és az entitások egymástól függetlenül léteznek. A Form feladata, hogy szükség esetén az Edit-ek és az entitások adatait szinkronba hozza. Míg az AssociationEdit-ek már maguk elvégzik az entitások módosításait.

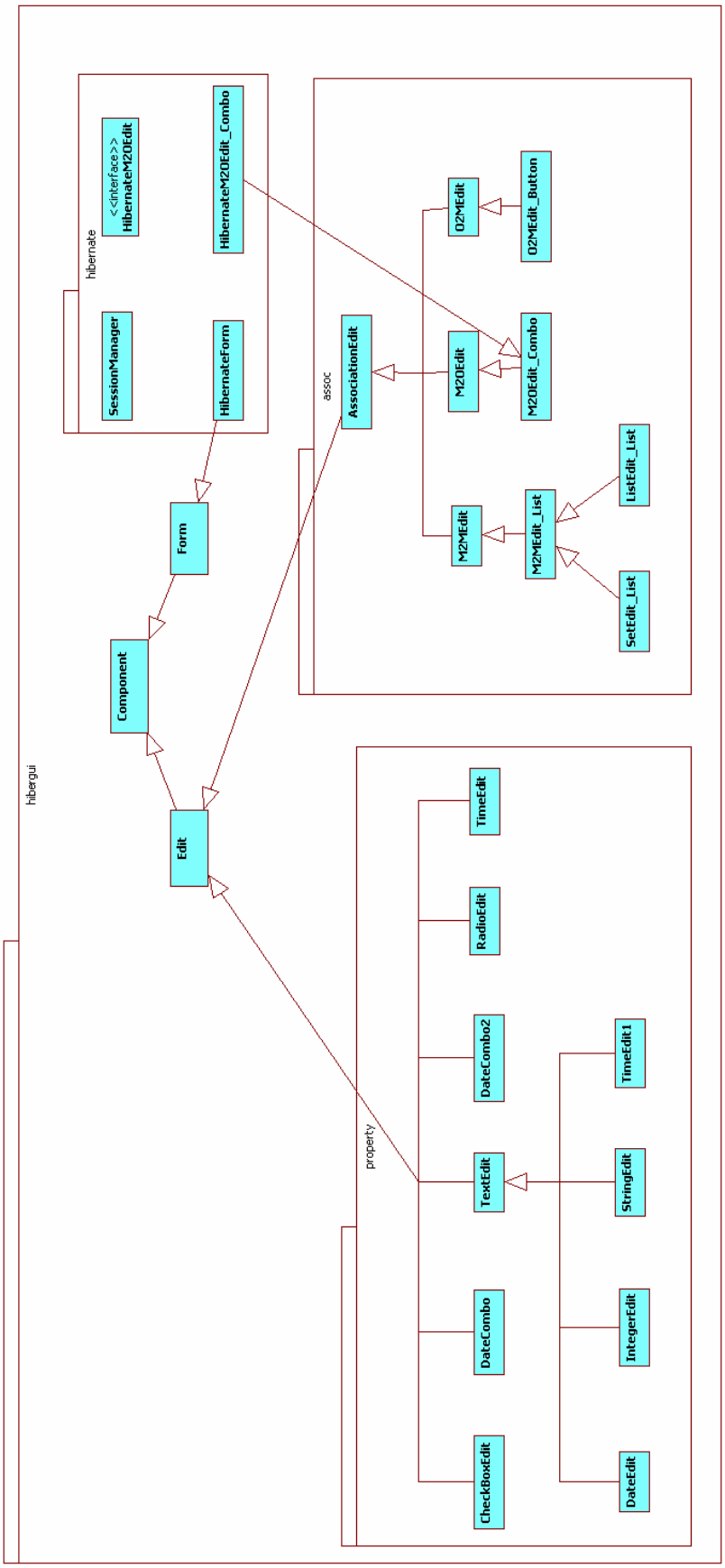
HibernateForm

Mivel a From-ban az adatok kezelését végzõ metódusok implementációja hiányzik, ezért szükség van egy a perzisztencia kezelést megvalósító leszármazottra. A HibernateForm közvetlenül a Form-ból származik és fenti metódusokat a Hibernate segítségével megvalósítja.

A HibernateForm és a Hibernate-hoz kapcsolódó egyéb osztályok a hibergui.hibernate csomagban vannak definiálva. A HibernateForm tartalmaz egy Hibernate Session példányt, amin keresztül éri el a Hibernate szolgáltatásait. A Session példány beállítását az ugyanezen csomagban lévõ SessionManager osztályon keresztül tehetjük meg.

SessionManager Feladata a Hibernate megfelelõ bekonfigurálása és a Hibernate Session példány tárolása. A Session példányt a getSession() metódussal érhetjük el. Az egyke tervezési mintát követve eléri, hogy alkalmazásonként csak egy példány lehet belõle és, hogy bárhonnan könnyedén elérhetõ legyen.[4]

1. ábra Componenet-ek osztályhierarchiája



3.4 A tesztalkalmazás

A keretrendszer lehetõségeinek és mûködésének felderítésére rendelkezésre áll a Zemplén Maraton nevû alkalmazás. Ez egy kerékpáros maraton nevezéseit és eredményeit kezelõ szoftver.

Az adatmodell

Az entitásokat Java bean-ek alkotják, melyeket Hibernate Mapping XML állományokkal képezünk le az adatbázis relációs modelljébe. A Hibernate beállítását a Hibernate Konfigurációs állományokon keresztül végzi.

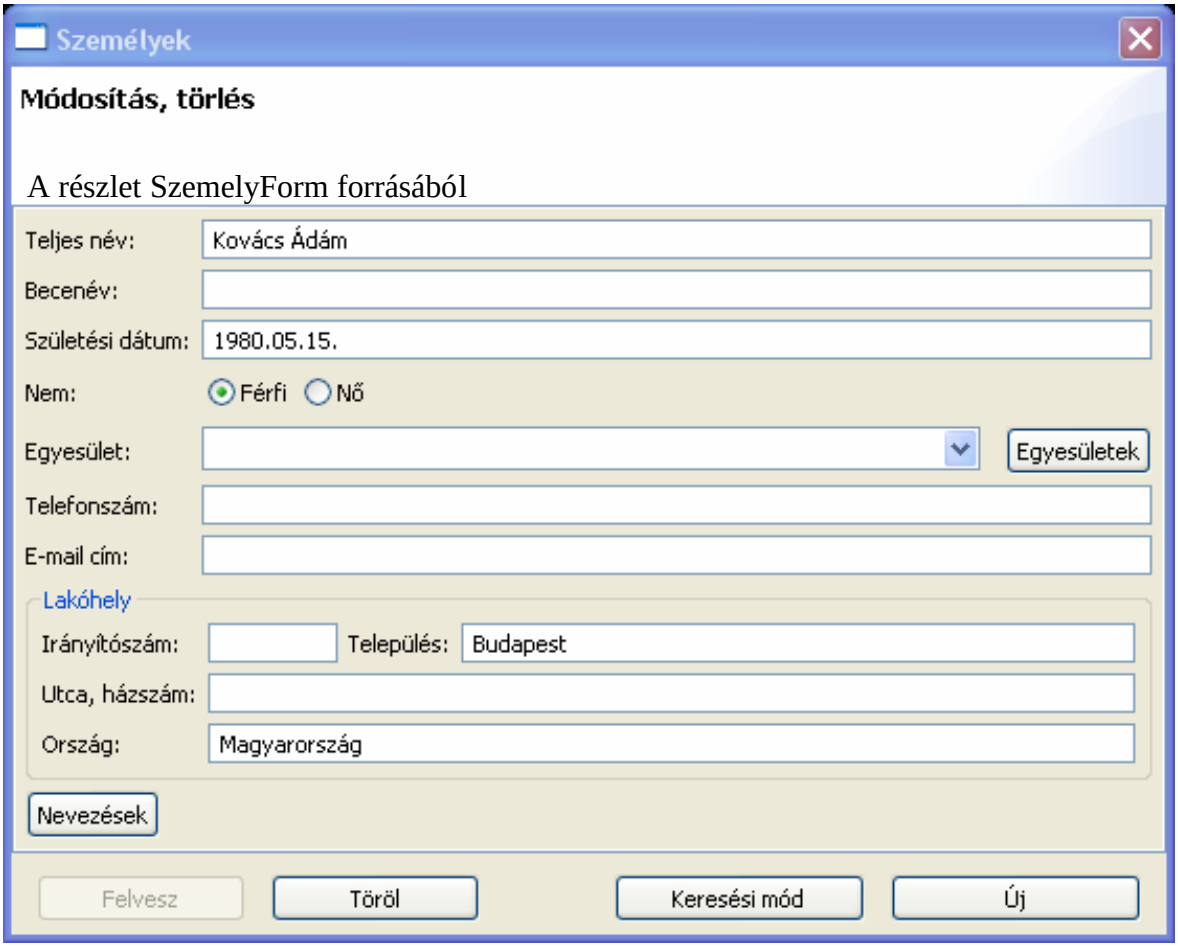
Az adatbázis

Egy pehelysúlyú, memóriában dolgozó HSQLDB példány alkotja az adatbázist. Az adatbázis kezelõ egyetlen darab jar-ból áll, telepítést nem igényel,. Magát az adatbázist pedig egy SQL script tartalmazza. Az adatbázis kezelõ indításakor a script alapján a memóriában létrehozza és feltölti a táblákat. A munka a memóriában történik, aminek a tartalma leállításakor vissza lesz írva a script-be.

HiberGUI elemek

Az egyes egyedtípusokhoz létre vannak hozva az õket kezelõ Form leszármazottak. Egy Form-ban megadjuk az Editeket, azt hogy milyen jellemzõket kell beállítaniuk. Itt határozzuk meg a Form elemeinek az elrendezését, és a különféle címkéket is. A From-ok mellett az AllomasCombo osztályból is vannak saját leszármazottaink.

2. ábra A SzemelyForm felülete



```
public class SzemelyForm extends HibernateForm<Szemely> {


    StringEdit eTeljesNev, eBecenev;
    DateEdit eSzuletesiDatum;
    RadioEdit<Character> eNem;
    EgyesuletCombo eEgyesulet;
    StringEdit eTelefonszam;
    StringEdit eEmail;
    StringEdit eLakcimIrszam, eLakcimTelepules, eLakcimUtca, eLakcimOrszag;
    O2MEdit_Button<Nevezes> eNevezesek;

    public SzemelyForm(Composite cptParent, int mode) {
        super(cptParent, mode);

        setWindowTitle("Személyek");

        eLakcimOrszag.setDefault("Magyarország");

    }

    @Override
    protected Composite createFields(Composite cptParent) {
        Composite cpt = new Composite(cptParent, SWT.BORDER);

        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        cpt.setLayout(gridLayout);

        GridData data;


        new Label(cpt, SWT.NONE).setText("Teljes név:");
        eTeljesNev = new StringEdit(this, cpt, NOT_NULL, "nev", "teljes név");
        data = new GridData(GridData.FILL_HORIZONTAL);
        eTeljesNev.setLayoutData(data);


        new Label(cpt, SWT.NONE).setText("Becenév:");
        eBecenev = new StringEdit(this, cpt, NONE, "becenev", "becenév");
        data = new GridData(GridData.FILL_HORIZONTAL);
        eBecenev.setLayoutData(data);
```

```
new Label(cpt, SWT.NONE).setText("Születési dátum:");
eSzuletesiDatum=new DateEdit(this,cpt,NONE,"szuletesiDatum","születési      dátum");
data = new GridData(GridData.FILL_HORIZONTAL);
eSzuletesiDatum.setLayoutData(data);


new Label(cpt, SWT.NONE).setText("Nem:");
eNem = new RadioEdit<Character>(this, cpt, NOT_NULL, "nem", "nem") {
    @Override
          protected String[] createLabels() {
        return new String[]{"Férfi", "Nő"};
    }
};
```

4 Fejlesztés elõkészítése

Az elõzõ fejezetben a HiberGUI célját, szerkezetét és mûködését ismertettem. Ebben a fejezetben a perzisztencia kezelés és néhány implementációs részletrõl lesz szó.

4.1 A grafikus elemek és a perzisztencia kezelés

Az MVC minta megköveteli, hogy az adatok perzisztens kezelése és azok megjelenítése különválasztva történjen. Ezt a szabályt a HiberGUI következetesen be is tartja. Míg a komponensek õsosztályában, a Component-ben vannak deklarálva az alapvetõ perzisztencia kezelési mûveletek, addig a komponensek leszármazottjaiban vannak ezek implementálva, ott ahol már a felhasználói felülettel kapcsolatos mûveletek nincsenek. A megjelenítés és az adatkezelés kódja nem keveredik. Egyedül egy probléma van ezzel a megközelítéssel: Java-ban az osztályhierarchia statikus. Ennek következtében nem tehetjük meg, hogy a HiberGUI forráskódjainak módosítása nélkül kicserélhessük a perzisztencia kezelést implementáló részt. Ha szeretnénk alkalmazást készíteni, vagy kiegészíteni a HiberGUI-t akkor az egyes entitások kezeléséhez a Form osztályt kell származtatni. A szükséges perzisztencia kezelõ implementációk a Form-ban nincsenek jelen, ezért vagy nekünk kell ezeket implementálni, vagy már egy, a Hibernate-on alapuló implementációból kell származtatni, ami a HibernateForm. A problémát az okozza, hogy ha más perzisztencia kezelõ implementációra szeretnénk váltani, de módosítás nélkül akarjuk megtartani az elkészített komponenseinket, akkor ezt a jelenlegi rendszerben nem tehetjük meg. Ugyanis a HibernateForm-ból vagy egyéb implementációból való származtatás után az õsosztályt a forráskód módosítása nélkül nem tudjuk a gyerekekben módosítani.

A megoldás az, hogy a komponenseket és a perzisztencia megvalósító osztályt nem öröklõdéssel, hanem kompozícióval rendelem egymáshoz. A következõ módosításokat végeztem el:

- Form absztrakt metódusai alapján létrehoztam egy perzisztencia kezelést végzõ interfészt. Ez lett a FormPersister interfész.
- A Form osztályban létrehoztam egy FormPersister típusú mezõt, és a megfelelõ beállító és lekérdezõ metódust hozzá.
- A Form perzisztencia kezelõ metódusainak összes elõfordulását felkutattam és a hívásaikat kicseréltem az FormPersister példány megfelelõ metódus hívásaira.

- Töröltem a Form perzisztencia kezelését végzõ metódusainak deklarációit.
- A elkészítettem FormPersister interfészt Hibernate segítségével implementáló osztályt. Ez lett a HibernateFormPersister osztály.
- A HibernateForm-ból származtatott osztályokat a Form-ból származtattam és gondoskodtam róla, hogy példányosítás után a FormPersister mezõ értéke egy HibernateFormPersister példány legyen.
- Töröltem a feleslegessé vált HibernateForm osztályt.

Ezután a teszt alkalmazás segítségével teszteltem az elvégzett kódátszervezés sikerességét, és hibákat javítottam.

A módosításoknak köszönhetõen, a perzisztencia kezelés implementációja még lazábban csatolódik a komponensekhez, így lehetõvé téve az implementáció könnyû cseréjét. Ehhez csak implementálni kell a FormPersister interfészt, és annak egy példányát meg kell adni a konkrét Form leszármazottaknak. Ezen kívül lett egy járulékos haszna is a módosításnak. Eddig a Formban deklarált perzisztencia kezelõ metódusok leszármazottakban látható bezárási szinttel rendelkeztek. Ez szükséges volt, mert csak így lehetett õket felüldefiniálni az implementációjukkal. Mivel Java-ban a bezárási szintje egy metódusnak nem csökkenthetõ, ezért ezek a metódusok a Form összes leszármazottjában láthatók voltak, miközben csak és kizárólag a Form eseményvezérlõ logikája használta õket. A módosítás után már lehetõvé vált a perzisztencia kezelõ mûveletek teljes elrejtése a Form leszármazottjai elõl.

A HibernateForm persister interfész forrása:

```
package hibergui.persister;

import java.util.List;

public interface FormPersister<T> {

     void commit();

     void rollback();

     void save(T entity);

     void update(T entity);

     void delete(T entity);

     List<T> findSuch();

     void refresh(T entity);
}
```

A T típus adja meg, hogy milyen típusú entitásokat kezelhet a FormPersister.

4.2 Apróbb módosítások a keretrendszeren

A késõbbi fejlesztések elõkészítése céljából a keretrendszer fõ osztályainak átvizsgálását és szükség esetén átszervezését végeztem ebben a pontban.

- Logikailag az alaposztályokba tartozó metódusokat kiemeltem a leszármazottakból.
- Ahol lehetett, szûkítettem a tagok láthatóságán.
- A nem használt metódusokat, fölösleges kódrészleteket töröltem.
- Szülõosztály mezõinek közvetlen elérését metódushívásra cseréltem.
- Az átláthatóbb osztály hierarchia érdekében létrehoztam az attribútumokat beállító egyedeknek egy közös õsosztályt, a PropertyEdit-et. Az asszociációkat beállító komponenseknek már AssociationEdit néven volt közös õsosztálya, így õket nem kellett módosítanom. Egységessé tettem az M2MEdit, M2OEdit és az O2MEdit leszármazottjainak neveit. Az új és a régi nevek összehasonlítása a 6. táblázatban

található. Ezeket az osztályokat a HiberGUI felhasználói fogják használni ezért fontos az egységes elnevezés és az egyértelmûség. Ezért négy szónál többõl nem állhat az osztályok azonosítója és a beállítandó kapcsolat számossága a névbõl egyértelmûen derüljön ki. Ezért a leszármazottak azonosítójának formátuma elõször a számossággal kezdõdik, utána következik egy utalás a komponens felhasználói felületére, végül, ha szükséges, akkor a From szó után a kapcsolódó egyedek tárolására szolgáló kollekció nevével végzõdik.

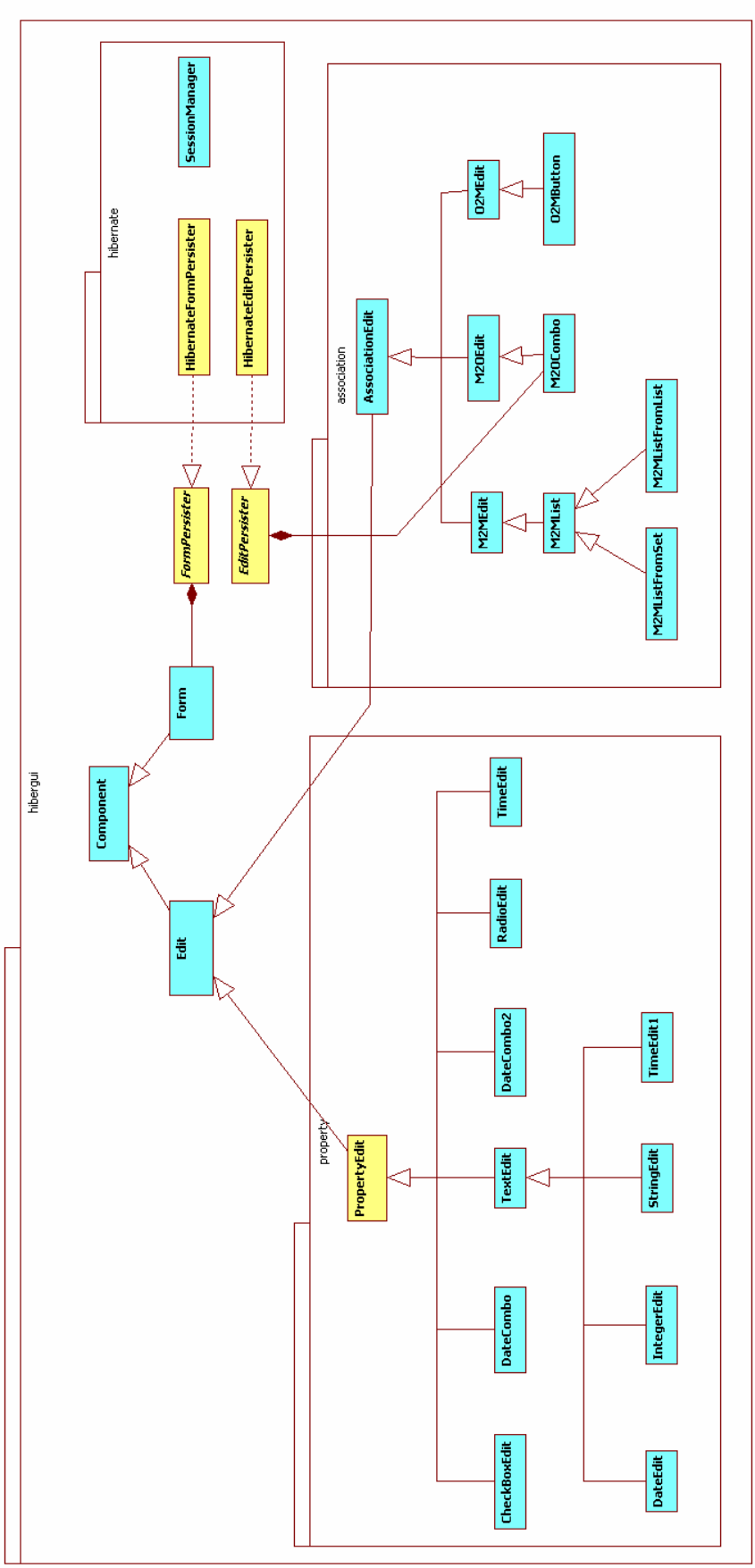
6. táblázat

Régi név	Új név	Jelentés
`M2MEdit_List`	`M2MList`	A felhasználó az elemeket egy listából választja ki.
`ListEdit_List`	`M2MListFromList`	A felhasználó az elemeket egy listából választja ki és a kapcsolat reprezentálása lista kollekcióval történik.
`SetEdit_List`	`M2MListFromSet`	A felhasználó az elemeket egy listából választja ki és a kapcsolat reprezentálása lista halmaz történik.
`M2OEdit_Combo`	`M2OCombo`	A felhasználó az elemeket egy legördülõ menübõl választja ki.
`O2MEdit_Button`	`O2MButton`	A felhasználó egy gomb megnyomásával tudja a kapcsolatot beállító komponenst megjeleníteni.

Az átalakítás után a tesztalkalmazással teszteltem a módosított elemeket.

A módosított keretrendszer osztálydiagramjának részlete a 3. ábrán található. Az új osztályok sárgával vannak jelölve.

3. ábra a módosított osztályok



5 Kapcsolatok

5.1 Az egyed kapcsolatokról

Az egyedek kapcsolatainak reprezentációja máshogy történik a grafikus felületen és máshogy a Java objektumok közt. A felhasználói felület az objektumoknál elvontabban kezeli a kapcsolatokat. A következõkben felsorolom a HiberGUI esetén szóba jöhetõ kapcsolatokat a számosság szerint csoportosítva. Mivel a HiberGUI csak bináris kapcsolatokat kezel, ezért elég csak ezeket tárgyalni. Ezek szerint megkülönböztetünk 1:1, 1:n és n:m számosságú kapcsolatot. Megkülönböztetjük a kapcsolat és a kapcsolat típus jobb és bal oldalát. Eszerint még beszélhetnénk n:1 számosságról is, de ez most egyenértékûnek tekinthetõ egy olyan 1:n számosságú kapcsolattal, aminek a két oldalát felcseréltük.

Azonos számosságnál a grafikus felületen megjelenõ kapcsolathoz bármely Java szintû reprezentáció tartozhat. A két megjelenés ortogonális egymásra.

Példa: Anya-gyermek viszony. Egy édesanyának több gyereke lehet, egy gyereknek pontosan egy édesanyja. Így a számosság 1:n. Ezt Java-ban ábrázolhatjuk úgy, hogy mind az anyához, mind a gyerekhez létrehozunk egy osztályt, és az anya osztályban deklarálunk egy gyerekeket tartalmazó kollekció típusú tagváltozót. Ez fogja az édesanyához tartozó gyerekeket nyilvántartani. Tegyük fel, hogy a gyereknek valamilyen okból nem deklaráljuk az édesanyját tároló tagváltozót, ebben az esetben nem fogja tudni ki az édesanyja. Ezzel szemben a felhasználói felületen igény lehet arra, hogy tudjuk gyerekhez hozzárendelni édesanyát, ne csak édesanyához rendelhetünk gyerekek halmazát.

A következõ csoportosításokban a Java entitások és a grafikus felületen megjelenõ egyedek kapcsolatainak lehetséges kombinációit adom meg. Ezek mind olyan kapcsolat reprezentációk ahol a kapcsolatok reprezentálásában csak a két résztvevõ egyed vehet részt, vagyis külön kapcsoló objektumot nem hozunk létre.

Az 1:1 számosságú kapcsolat

Java entitások szintjén:

a) A bal egyed referenciával rendelkezik a kapcsolat másik felére. Kapcsolat beállításakor a referenciát hordozó egyednek meg kell adni a másik egyedet és a szerepkört.

b) A jobb egyed referenciával rendelkezik a kapcsolat másik felére. Kapcsolat beállításakor a referenciát hordozó egyednek meg kell adni a másik egyedet és a szerepkört.

c) Mind a jobb oldali, mind a bal oldali egyed rendelkezik a másik félrõl egy referenciával. Ekkor az a) és a b) eset közösen igaz. A kapcsolat beállításakor mindkét referenciát be kell állítani.

Grafikus felület szintjén:

a) Bal egyedhez hozzárendeljük a másik egyedet.

b) Jobb egyedhez hozzárendeljük a másik egyedet.

Az 1:n számosságú kapcsolat

Java entitások szintjén:

a) A kapcsolat jobb oldalán résztvevõ fél referenciával rendelkezik a kapcsolt bal oldalán lévõhöz. Kapcsolat beállításakor a referencia hordozó egyednek meg kell adni a másik egyed referenciáját és a szerepkört.

b) A kapcsolat bal oldalán levõ fél referenciák egy kollekciójával bír. E kollekció tagjai reprezentálják a másik oldallal való viszonyt. Kapcsolat beállításakor e kollekció elemein kell mûveleteket végezni.

c) A bal oldali egyed a jobb oldali egyedek kollekciójával, míg a jobb oldali egyed a bal oldali egyed egy referenciájával rendelkezik. Ekkor az a) és a b) eset közösen igaz. Kapcsolat beállításakor mindét referenciát be kell állítani

Grafikus felület szintjén:

a) A bal oldali egyedhez kiválasztunk tetszõleges számú jobb oldali egyedet.

b) A jobb oldali egyedhez kiválasztunk egy bal oldali egyedet.

A rögzített sorrend ellenére az n:1 és 1:n kapcsolat egymással egyenértékûnek tekintjük, mert egyik kifejezhetõ a másikból.

Az n:m számosságú kapcsolat

Java entitások szintjén:

A kapcsolat jobb oldalán lévõ egyed bal oldali egyedek kollekciójával rendelkezik, és a bal oldali egyed is rendelkezik, egy a jobb oldali elemeket nyilvántartó kollekcióval. Itt minden esetben megegyeznek az entitás osztályok szerkezetei, az eltérések abból adódnak, hogy ha két egyed közt fennáll a kapcsolat, akkor kölcsönösen egymás kollekciójába kerülnek vagy sem. Eszerint:

a) Csak jobb oldali egyed kollekciójába kerül be a bal oldali, fordítva nem.

b) Csak a bal oldali egyed kollekciójába kerül be a jobb oldali, fordítva nem.

c) A jobb oldali egyed kollekciójába bekerül a bal oldali, és a bal oldali egyed kollekciójába bekerül a jobb oldali egyed.

Vagyis a kapcsolat beállításakor vagy a bal oldali egyed kollekciójához adjuk hozzá a jobb oldalit, vagy a jobb oldali egyed kollekciójához adjuk hozzá a bal oldalit, vagy kölcsönösen mindkét egyed kollekciójába felvesszük a másikat.

Grafikus felület szintjén:

a) Jobb oldali elemhez hozzárendelünk tetszõleges számú bal oldali elemet

b) Bal oldali elemhez hozzárendelünk tetszõleges számú jobb oldali elemet.

A reprezentációs módok általánosítása

Az elõbbi pontokban tárgyalt Java objektumok közti kapcsolat reprezentációkból sokfélék lehetnek. Aszerint, hogy a kapcsolatban a melyik egyed, melyikre hivatkozik egyirányú jobbról balra mutató, egyirányú balról jobbra mutató, vagy kétirányú kapcsolatról beszélni.

Általánosítva bármely számosságú és irányú kapcsolat reprezentálható maximum kettõ darab egyirányú kapcsolat alapján.

Ez az egy irányú kapcsolat vagy jobbról balra, vagy balról jobbra mutató 1:1, vagy 1:n kapcsolat. Az egyedtípusok felcserélésével pedig a jobb elemrõl bal elemre irányuló kapcsolatból elõállítható a bal elemrõl jobb elemre irányuló kapcsolat is.

Ezzel eljutottunk oda, hogy egy egyirányú 1:n és egy egyirányú 1:1, jobbról balra mutató kapcsolattal az összes eddig felsorolt Java objektumok közti kapcsolat típus megadható.

A többes számosságot reprezentáló kollekciók

A kollekciók elsõdleges szerepe az egyeddel kapcsolatban lévõ egyedek nyilvántartása. Emellett járulékos információkat is tárolhatnak a asszociációval kapcsolatban. A kollekciónak a szerepe csak a két egyed közti kapcsolat fennállásának eldöntésére szolgál. Vagy fennáll a kapcsolat vagy nem. Egyéb információval nem szolgál.

A kollekció a kapcsolatról egyéb információkat is tartalmaz. A résztvevõ egyedeknek sorrendje van. Eszerint lekérdezhetõ hogy az adott elem hányadik elem a kapcsolatban. Az elõzõ változat felfogható egy olyan elem sorozatnak, ahol az elemek sorszáma nem lényeges.

A résztvevõ elemeknek kulcsokkal vannak azonosítva a kapcsolatban. Az elõzõ változat felfogható egy olyan kulccsal rendelkezõ kapcsolatnak, ahol az kulcs típusa természetes szám, és ez adja az egyed sorrendjét. Itt ki kell kötni, hogy a kulcsoknak folytonosnak kell lennie.

5.2 Kapcsolatok beállítása a HiberGUI-ban

Mivel a HiberGUI egy általános célú keretrendszer ezért elvárjuk tõle, hogy az egyedek közti kapcsolatok beállítására is adjon egy általános megoldást. Itt követelmény az újrafelhasználhatóság és az egyszerû használhatóság. Az 5.1 alfejezetben láthattuk, mennyire sokféle módja lehet a kapcsolatok reprezentálásának és láthattuk hogyan lehet egyszerûbbé és újrafelhasználhatóságra alkalmasabbá tenni a kapcsolatok sokaságát.

A Java bean-ek kapcsolatait az AssociationEdit-ekkel lehet beállítani. Ezek, és a kezelhetõ kapcsolat viszonyait a következõ táblázatok tartalmazzák.

2. táblázat

AssociationEdit	Kapcsolat számossága	Reprezentáció Java szinten
`M2MEdit`	n:m	A bal oldali elem a jobb oldali elemek kollekcióját tartalmazza és a jobb oldali elem a bal oldali elemek kollekciójával rendelkezik.

M2OEdit	n:1 vagy 1:1	A bal oldali elem a jobb oldali elem referenciáját tartalmazza és a jobb oldali elem a bal oldali elemek kollekciójával vagy bal oldali elem referenciájával rendelkezik.
O2MEdit	1:n	A bal oldali elem a jobb oldali elemek kollekcióját tartalmazza és a jobb oldali elem a bal oldali elem referenciájával bír.

Az AssociationEdit az 2. táblázatban felsorolt három közvetlen alosztállyal rendelkezik. Ezek absztrakt osztályok. Céljuk számosság szerint csoportosítani az AssociationEdit-eket, és kiemelni a közös sajátosságaikat.

3. táblázat

Osztály neve	**Megjelenés a grafikus felületen**
M2MList	A jobb oldali elemek listájából választhatunk.
M2MListFromList	A jobb oldali elemek listájából választhatunk.
M2MListFromSet	A jobb oldali elemek listájából választhatunk.

A 3. táblázat tartalmazza az n:m számosságot beállító M2MEdit-ek leszármazottjait. A M2MList két leszármazottja M2MListFromList és a M2MListFromSet.

4. táblázat

M2OEdit	**Megjelenés a grafikus felületen**
M2OCombo	Jobb oldali elemeket tartalmazó legördülõ menü. Csak egy elem választható.

A 4. táblázat tartalmazza az 1:n kapcsolatok beállítását végzõ M2OEdit leszármazottja.

5. táblázat

O2MEdit	**Megjelenés a grafikus felületen**
O2MButton	Egyetlen gomb.

A 5. táblázat tartalmazza az O2MEdit leszármazottját, melyek az 1:n kapcsolatok beállításáért felelõs.

A táblázatokból látszik, hogy a grafikus komponensek felülete és az hogy milyen módon vannak reprezentálva a Java bean-ek közti kapcsolatok összefügg. Ezeket a függõségeket érdemes lenne feloldani.

5.3 A kapcsolatok beállításának továbbfejlesztése

A cél a HiberGUI olyan módon való áttervezése és implementálása, hogy a kapcsolatok kezelése általánosabb és kevesebb fejlesztõi munkát kívánó legyen.

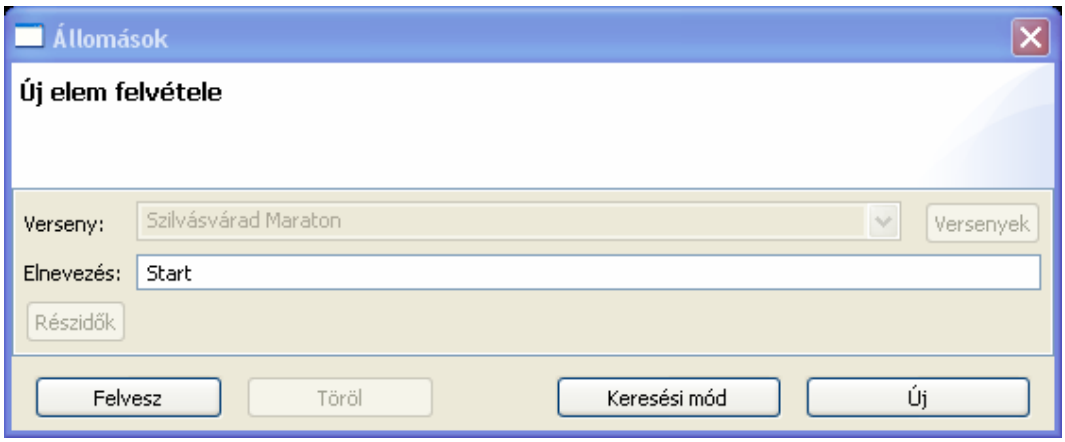
Azt szeretnénk elérni, hogy a kapcsolat számosságának és a résztvevõ entitás típusok megadása után a fejlesztõnek a felhasználói felület oldaláról tekintve ne kelljen az entitások közti kapcsolatok konkrét Java megvalósítását ismernie.

Egy konkrét probléma

A tesztalkalmazáson keresztül bemutatok egy nehézséget, ami a HiberGUI-val való munka során elõfordulhat.

Adott az Allomas és a Verseny entitás esete, melyek között kétirányú, n:1 számosságú kapcsolat van. Az Allomas a verseny nevû mezõben tárolja a hozzá tartozó állomás példányát, míg a Verseny az allomasok nevû mezõ halmazában tárolja az állomásait.

4. ábra AllomasForm kinézete



Részlet az állomás forrásából:

```
public class Allomas  implements java.io.Serializable {

     private Integer id;
     private String elnevezes;
     private Verseny verseny;
     private Set<Reszido> reszidok;
```

Részlet a Verseny forrásából:

```
public class Verseny  implements java.io.Serializable {

     private Integer id;
     private String elnevezes;
     private String helyszin;
     private Date idopont;
     private Set<Nevezes> nevezesek;
     private Set<Tav> tavok;
     private Set<Kategoria> kategoriak;
     private Set<Allomas> allomasok;
```

Mindkét osztály összes tagváltozójához meg vannak írva a szabványos beállító és lekérdezõ metódusok, amikkel szabadon kezelhetjük a privát tagok értékeit.

Ahogy az 5. ábrán látszik, az AllomasForm segítségével egy adott versenyhez, itt a Szilvásvárad Maratonhoz tudunk új állomást készíteni és felvenni. Amikor a létrehozunk egy új állomást az elnevezéshez tartozó StringEdit String típusú értéke és a VersenyCombo Verseny típusú értéke az AllomasForm extract(Allomas) metódusa által beállítja a nevet és a versenyt. A probléma ott jelentkezik, hogy ez a kapcsolat kétirányú, így külön hozzá kell adni az állomás versenyének allomasok nevû halmazához az új állomást. Ezt pedig magának a programozónak kell megtennie, mielõtt az állomás adatbázisba való mentésre kerül. Vagyis felül kell definiálnia a save(Allomas) metódust a FormPersister-ben. A korábbi rendszerben pedig az AllomasForm-nak kellett felüldefiniálni a save(Allomas) metódusát. Ha ezt nem tennénk meg, akkor a Hibernate nem tudná az egyed kapcsolatát leképezni az adatbázisba.

Az asszociator csomag

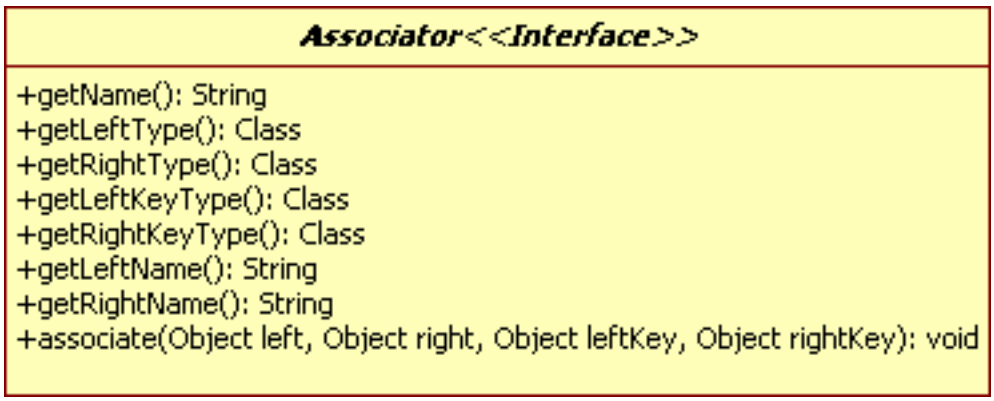
Ha sikerülne elérni, hogy már a kapcsolat beállításakor bináris kapcsolat esetén mindkét oldal állítódjon be, akkor nem lenne szükség a mentés elõtti kézi beavatkozásra. Ennek megvalósítására hoztam létre az associator csomagot. Ez tartalmazza azokat az interfészeket és osztályokat melyek segítségével a kapcsolatok beállítása már mentés elõtt helyesen történik.

A reprezentációs módok általánosítása és **A többes számosságot reprezentáló kollekciók** pontban tárgyaltak alapján egy elkészítünk egy általános, egy irányú 1:n és egy általános egy 1:1 kapcsolatot beállító eszközt, és a többes számosságú kapcsolatok kollekcióit kulcs értékek alapján kezeljük, akkor tetszõleges kapcsolatot be tudunk állítani egy, esetleg két ilyen eszköz segítségével. Az így kapott eszközt pedig az AssociationEdit-ekhez hozzárendelve fel tudjuk arra használni, hogy az Edit-et befogadó Form megfelelõen állítsa be a kapcsolatokat az entitások közt.

Az Associator intefész

Az Asszociator interfészen keresztül tudunk két Java bean között kapcsolatot létrehozni. Egy asszociátorral mindig csak egy fajta kapcsolat típust lehet beállítani.

6. ábra



Az asszociátor példányosítás után nem módosítható tartalmú. Ezt jelzi az, hogy csak lekérdezõ metódusai vannak a jellemzõihez, ami a 6. ábrán lévõ osztály diagrammon látható. Az interfész lényegét az associate metódus adja. Meghívásával lehet a bal oldali és a jobb oldali egyed közt az asszociátorhoz rendelt kapcsolatot beállítani. Az elsõ paraméter a bal, a második a jobb oldali egyed. A harmadik paraméter a bal oldali egyed kapcsoló

kollekciójának kulcsértéke, a negyedik pedig a jobb oldali egyedben lévõ kapcsoló kollekció kulcsának értéke. Ezekkel lehet megadni sorrendet, vagy kulcs értéket a kapcsolat jellegétõl függõen. Az utolsó kettõ paraméter csak többes számosságú kapcsolatok esetén adható meg, különben null értéket kell megadni.

Hogy futás közben létezõ metaadatokból is lehessen asszociátort létrehozni, ezért a paraméterek típusa nem lett kikötve. Szándékosan nem használtam generikus típust. Ezzel lemondtam a fordítás idejû típusellenõrzésrõl. Helyette futás idejû típusellenõrzést alkalmaztam. Az asszociátor paramétereinek típusát a getLeftType, getRightType, getLeftKeyType és getRightKeyType metódusokkal lehet lekérdezni.

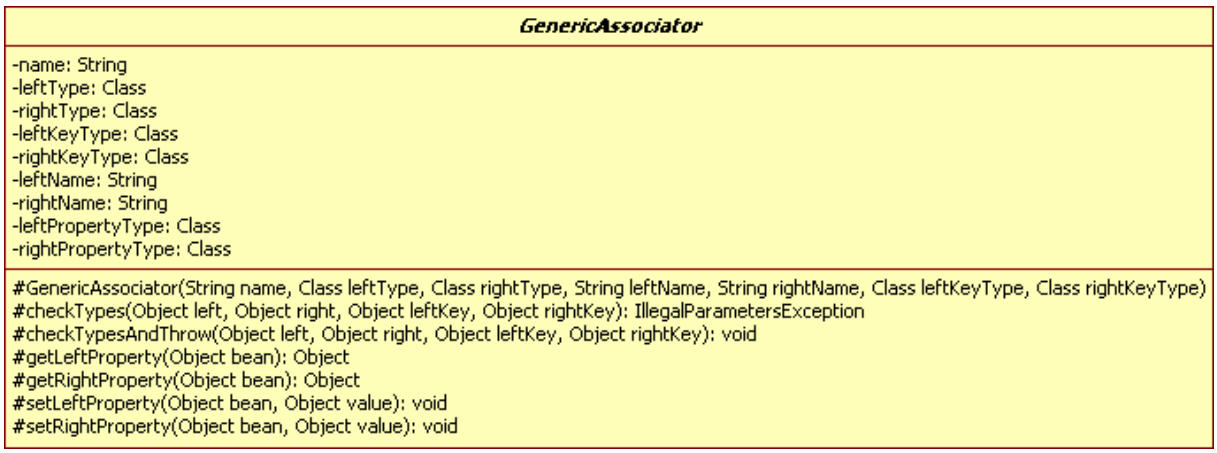
Azt, hogy a kapcsolatért, milyen nevû jellemzõ felelõs az egyes egyedekben a getLeftName és a getRightName metódusokkal lehet lekérdezni.

Végül a getName metódussal lehet megtudni, hogy mi a kezelendõ kapcsolatnak a neve.

A GenericAssociator

A GenericAsszociator az Associator interfész egy általános célú implementálása. Osztály diagramja a 7. ábrán látható. Absztrakt osztály, mivel nem implementálja az Associator associate metódusát. A lekérdezõ metódusait megvalósítja, de ez nincs feltüntetve a diagramon. A szigorú futásidejû típusellenõrzés ebben az osztályban van megvalósítva, a Java Reflection API segítségével. A típusellenõrzést a leszármazottjainak kell elvégeznie az associate metódus aktuális paraméterein. Erre két segédmetódus is szolgál, az egyik a checkTypes, a másik a checkTypesAndThrow. Mindkettõ ugyanúgy ellenõrzi az átadott paramétereket, a különbség abban van, hogy a checkTypesAndThrow hiba esetén kivételt dob, addig a checkTypes, visszatér a kivétellel, de nem dobja el. Bizonyos helyzetekben az utóbbi módszer használata kényelmesebb. A bal vagy a jobb oldali entitás kapcsolatért felelõs tagváltozóinak kezelésére szolgál a getLeftProperty, getRightProperty, setLeftProperty, setRightProperty metódus.

7. árba



Egy irányú kapcsolatokat beállító asszociátorok

Az ide tartozó asszociátorok konstruktorának formális paraméter listája String name, Class leftType, Class rightType és String leftName elemekbõl áll.

Vagyis a kapcsolat nevének, és a résztvevõ egyedtípusok megadása után csak a bal oldali egyed egy jellemzõjének nevét kell megadni. Ezen fog alapulni a kapcsolat.

ToOneAssociator: Egyes számosságú, egyirányú kapcsolat. A leftName nevû jellemzõ a baloldali egyedet fogja tárolni. Az associate metódus hívásakor mindkét kulcshoz null értéket kötelezõ megadni.

ToManyListAssociator: Többes számosságú, listán alapuló egyirányú kapcsolat. A leftName nevû jellemzõ a jobboldali egyedeket tartalmazó listát tárolja. Így a kapcsolatban az elemek sorrenddel bírnak. Az indexet a leftKey-el lehet megadni. A rightKey-nek null-nak és az indexnek Integer típusúnak kell lennie.

ToManySetAssociator: Többes számosságú, halmazon alapuló egyirányú kapcsolat. A rightName nevû jellemzõ a jobboldali egyedeket tartalmazó halmazt tárolja. Mindkét kulcsnak null értékûnek kell lennie.

Egyéb asszociátorok

EchoAsszociator

Ez az egyetlen asszociátor amely nem a GenericsAssociator-ból van származtatva. Ennek oka a nevében rejlik: Feladata hibakeresés céljából naplózni metódusainak hívását és a kapott paramétereket.

ReverseAssociator

Felcseréli egy asszociátor jobb és bal egyedtípusát.

Konstruktora paraméterül várja az asszociáció nevét és a becsomagolandó asszociátor példányt.

BidirectionalAssociator

Kettõ darab egy irányú kapcsolatot beállító asszociátorból képez egyet. A konstruktora várja a nevet, és a két asszociátor példányt. A másodiknak fordított irányúnak kell lennie. Segítségével tudunk konstruálni kétirányú 1:n kapcsolatot beállító asszociátort. Ehhez szükségünk van egy ToManySetAssociator-ra és egy ToOneAssociator-ra. Ez utóbbit a ReverseAssociatorral be kell csomagolni. Ezután a két asszociátor a BidirectionalAssociator-ral megkapjuk a kívánt asszociátort.

Az asszociátorok teljes osztálydiagramját a 8. ábra tartalmazza.

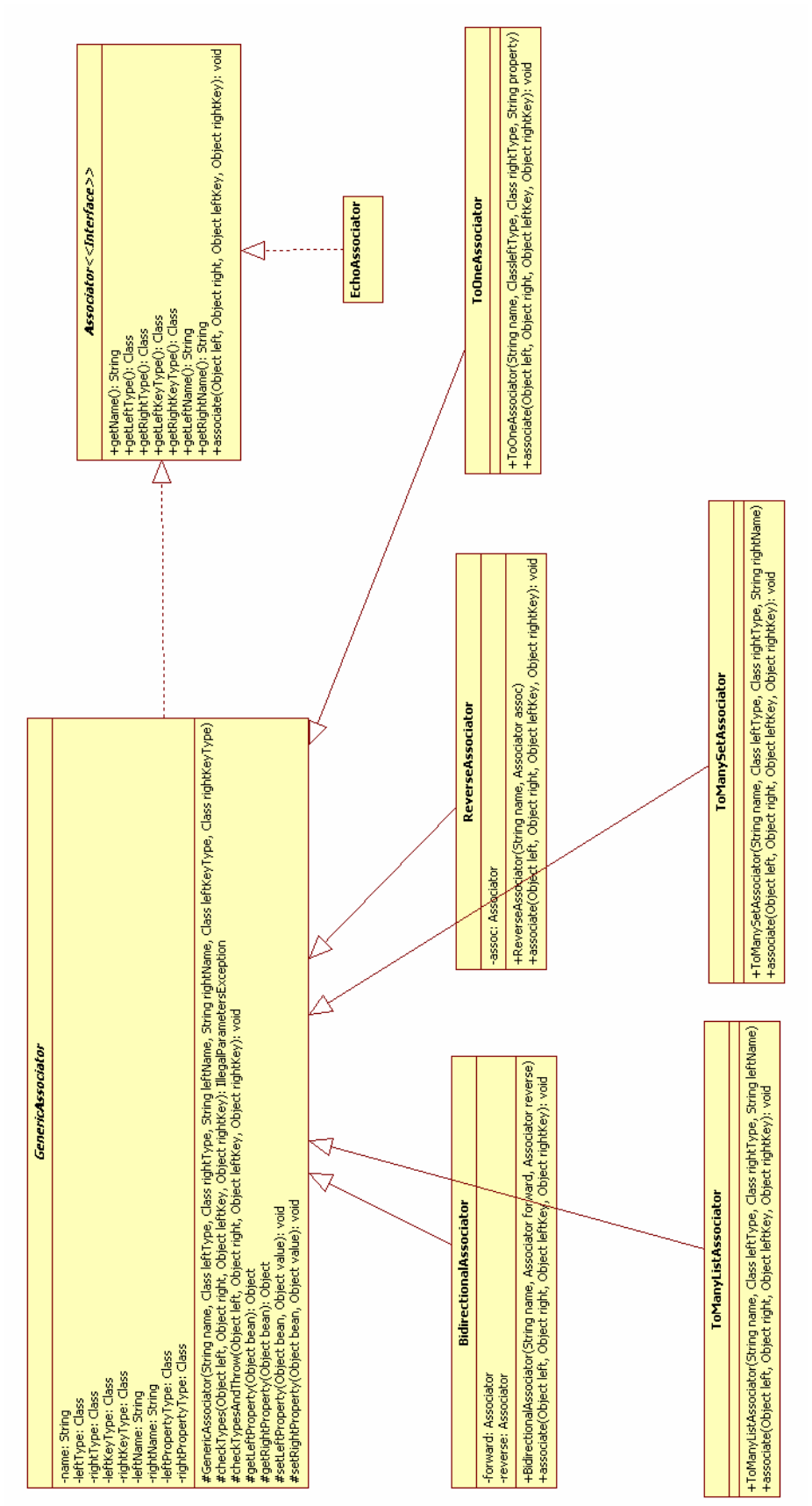
Az asszociátorok használata

Az objektum-relációs leképezés metaadataiból, a HiberGUI grafikus komponensek metaadataibõl és/vagy a fejlesztõ által explicite megadott adatokból minden egyes AssociationEdit-hez elkészítjük a megfelelõ Association példányt. Ezt a rendszer konfigurációs idejében el lehet végezni. Az AssociationEdit-ekben egy mezõ tárolja az Associator példányt. Mikor az AssociationEdit-et tartalmazó Form az extract metódusával az Edit-ek tartalma alapján aktualizálja az entitás típusát, akkor két dolog fog lejátszódni. PropertyEdit esetén az Edit tartalmát csak le kell kérdezni és az entitás megfelelõ attribútumának értékül kell adni. AssociationEdit-ek esetén pedig az Associator példányt le kell kérdezni, és az AssociationEdit által tartalmazott entitásra, vagy entitásokra egyenként meg kell hívni az Associator associate metódusát. Ezen egyedek képezik az asszociációk jobb, míg a Form egyede az asszociáció bal oldalát.

Ennek az átalakításnak pedig nagy elõnye, hogy tetszõleges 1:n, 1:1, n:1 n:m számosságú és tetszõleges irányú kapcsolatokat is lehet kezelni a megfelelõ asszociátor összeállításával. Emellett más kollekciókkal is könnyedén megvalósíthatók a többes kapcsolatok, és ekkor ezek is újrafelhasználhatók.

A fejlesztés végén egy mélyreható átvizsgálásnak vetettem alá a rendszer. Ezután a példaalkalmazással teszteltem, próbálva minél többféle kapcsolatot és funkciót kipróbálni.

8. ábra


Associator<<Interface>>
+getName(): String
+getLeftType(): Class
+getRightType(): Class
+getLeftKeyType(): Class
+getRightKeyType(): Class
+getLeftName(): String
+getRightName(): String
+associate(Object left, Object right, Object leftKey, Object rightKey): void
EchoAssociator
ToOneAssociator
+ToOneAssociator(String name, ClassleftType, Class rightType, String property)
+associate(Object left, Object right, Object leftKey, Object rightKey): void
GenericAssociator
-name: String
-leftType: Class
-rightType: Class
-leftKeyType: Class
-rightKeyType: Class
-leftName: String
-rightName: String
-leftPropertyType: Class
-rightPropertyType: Class
#GenericAssociator(String name, Class leftType, Class rightType, String leftName, String rightName, Class leftKeyType, Class rightKeyType)
#checkTypes(Object left, Object right, Object leftKey, Object rightKey): IllegalParametersException
#checkTypesAndThrow(Object left, Object right, Object leftKey, Object rightKey): void
#getLeftProperty(Object bean): Object
#getRightProperty(Object bean): Object
#setLeftProperty(Object bean, Object value): void
#setRightProperty(Object bean, Object value): void
ReverseAssociator
-assoc: Associator
+ReverseAssociator(String name, Associator assoc)
+associate(Object left, Object right, Object leftKey, Object rightKey): void
ToManySetAssociator
+ToManySetAssociator(String name, Class leftType, Class rightType, String rightName)
+associate(Object left, Object right, Object leftKey, Object rightKey): void
BidirectionalAssociator
-forward: Associator
-reverse: Associator
+BidirectionalAssociator(String name, Associator forward, Associator reverse)
+associate(Object left, Object right, Object leftKey, Object rightKey): void
ToManyListAssociator
+ToManyListAssociator(String name, Class leftType, Class rightType, String leftName)
+associate(Object left, Object right, Object leftKey, Object rightKey): void

6 Összefoglalás

A szakdolgozatomon keresztül megismerkedhettünk egy készülõ keretrendszer mûködésével, felmerülõ hiányosságaival és azok pótlásával. Végighaladtunk a megismerés és tervkészítés, a kód átalakítás és minõségjavítás, a probléma feltárás és megoldás lépésein. Sikerült elkészíteni a rendszer egy részletesebb dokumentációját. Elõkészítettem a késõbbi fejlesztések számára a szoftvert, és egy mûködõképes, bevált megoldást találtam egy összetett problémára. Ez pedig továbbfejleszthetõbbé, általánosabbá és jobban kezelhetõvé tette a meglévõ rendszert.

7 Köszönetnyilvánítás

Szeretnék köszönetet nyilvánítani Espák Miklós témavezetõmnek szakértelméért, türelméért és az általa adott lehetõségekért.

Sipos István Balázsnak a sok segítségért és bíztatásért.

Családomnak a támogatásért és kedvességért.

8 Irodalomjegyzék

[1] Sun Microsystems: Java Platform, Enterprise Edition 5 Specification, 2006

[2] Sun Microsystems: JSR-000220 Enterprise JavaBeans v.3.0 Specification, 2006

[3] Christian Bauer, Gavin King: Java Persistence With Hibernate, Manning, 2007

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994