Tallinna Ülikool

Informaatika Instituut

Hibernate raamistiku õppematerjal

Seminaritöö

Autor: Lauri Elias

Juhendaja: Jaagup Kippar

Tallinn 2011

1 Sisukord

2 Sissejuhatus

Antud üliõpilastöö eesmärk on tutvustada Hibernate raamistiku võimalusi. Lühidalt tutvustatakse Hibernate-i alternatiive. Näited viiakse läbi NetBeans IDE-s, seega sisaldab töö ka seletusi viimase kasutamiseks. Hibernate-i kasutatakse tihti koos teiste raamistikega, näiteks Spring, vaatlusele tuleb ka kahe raamistiku võimaluste kombineerimine.

Autor on motiveeritud teemal kirjutama, kuna enamus kättesaadavatest õppematerjalidest on inglise keeles ja eestikeelsed väljaanded puuduvad sootuks ning kuna Hibernate on ülipopulaarne raamistik, mille tundmist paljud Java EE arendajaid otsivad tööandjad nõuavad. Näiteks 26.10.2011 seisuga oli CV-keskuse 109-st Infotehnoloogia valdkonda kuuluvast tööpakkumisest 3 Hibernate raamistiku tundmise nõudega ja sama raamistiku kasutamist nõuti Webmedia Suveülikooli kandideerivatelt üliõpilastelt.

Autor peab vajalikuks lisada, et ei püüa siinses töös kirjeldada kogu Hibernate raamistiku funktsionaalsust ja luua koodinäiteid iga võimaliku probleemi lahendamiseks kõigil võimalikel viisidel. Näiteks saab Hibernate-i funktsionaalsust kasutada nii annotatsioonide kui XML deklaratsioonifailide kaudu, kuid autor näitab ära vaid ühe variandi, kuivõrd Hibernate raamistiku dokumentatsioon on piisavalt põhjalik, et iga tähelepanelik lugeja suudab pärast koodinäite läbiproovimist ise internetist ülejäänud meetodite kirjeldused ja nüansid üles leida.

3 Võõrkeelsete lühendite loetelu

XML – *Extensible Markup Language*, laiendatav markeerimiskeel

POJO – *Plain Old Java Object*, harilik Java objekt

API – *Application Programming Interface*, rakendusliides ehk programmiliides (vallaste.ee, 2011)

JPA – *Java Persistence API,* Java püsiesituse API (Vilgota, 2003)

JDBC – *Java Database Connectivity*, Java andmebaasipöördus

DBMS – *Database Management System*, andmebaasihaldur

SQL – *Structured Query Language*, struktuurpäringukeel

HQL – *Hibernate Query Language*, Hibernate-i arendajate poolt väljatöötatud struktuurpäringukeel

ORM – *Object Relational Mapping*, objektimudeli kujutamine relatsiooniliseks andmemudeliks

IDE – *Integrated Development Environment*, integreeritud programmeerimis-keskkond

J2EE – *Java 2 Platform Enterprise Edition*, Java platvorm mitmekihiliste serverorienteeritud firmarakenduste jaoks

JAXB – *Java Architecture for XML Binding*, võimaldab Java rakenduse koostajal hõlpsalt siduda XML nimeruum Java arhitektuuriga (Karring, 2005)

JAR – *Java Archive*, Java arhiivfail

CRUD – *Create, Read, Update, Delete*, andmebaasioperatsioonid, vastavalt: loo, loe, uuenda, kustuta

AOP – *Aspect Oriented Programming*, aspekt-orienteeritud programmeerimine (Mäehans, 2011)

IoC – *Inversion of Control*, kontrolli ümberpööramine (Raudjärv, 2005)

DAO – *Data Access Object*, andmepääsu objekt

URL – *Uniform Resource Locator*, internetiaadress

4 Materjalid internetis

http://www.vaannila.com/hibernate/hibernate-tutorial/hibernate-tutorial.html

Põhjalik ja kergesti mõistetav inglisekeelne juhend.

http://www.roseindia.net/hibernate/

Põhjalik juhend, sobib hästi alustamiseks. Leheküljel on üsna palju reklaami. Inglise keelne.

http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html/

Ametlik juhend esimese Hibernate rakenduse realiseerimiseks. Väga korrektne, koos täieliku dokumentatsiooniga. Mitte nii lihtsasti jälgitav kui eelnevad juhendid. Inglise keelne.

http://netbeans.org/kb/docs/java/hibernate-java-se.html

Tasemel juhend Hibernate-i kasutamiseks NetBeans IDE-ga. Inglise keelne.

http://www.allapplabs.com/hibernate/introduction_to_hibernate.htm

Ülipõhjalik, mõnevõrra keeruline juhend. Inglise keeles.

5 Tutvustus

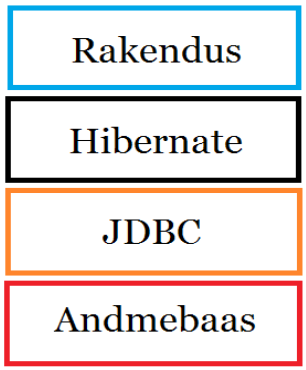
5.1 Mis on püsiesitatavus?

Ingl. k. persistence. Peaaegu kõik rakendused vajavad viisi andmeid jäädavalt salvestada. Java maailmas tähendab see üldjuhul andmete ülekandmist SQL-andmebaasi. Püsiesitatavus võimaldab objektil säilida ka pärast seda, kui protsess, mis ta tekitas, on lõppenud. (Bauer & King, 2005)

5.2 Mis on Hibernate?

Hibernate on avatud lähtekoodiga Java raamistik, mille eesmärk on arenduse hõlbustamine. Hibernate kasutab POJO mudelit, et kaardistada Java objekte relatsioonilisse andmebaasi, mille läbi on tagatud objektide püsiesitus. Hibernate aitab luua skaleeritavaid, kergesti porditavaid rakendusi, mille komponentide sõltuvust andmebaasitasandist ja teistest komponentidest on minimiseeritud. Lisaks vähendab Hibernate raamistik oluliselt rakenduse programmeerija poolt kirjutatava koodi pikkust – seda deklaratiivsete seadistusfailide abil, mis esinevad nii XML kui ka properties vormingutes. Hibernate toetab sama funktsionaalsust ka annotatsioonide kaudu. Lisaks on võimalik Hibernate-i seadistada ka programmaatiliselt. Hibernate raamistik laiendab JPA, JDBC, DBMS ja SQL võimalusi. (Seddighi, 2009)

Hibernate võib kujutleda lisa abstraktsioonikihina andmebaasiga suhtlevate draiverite ja Java rakenduse vahel. See tähendab, et arendatud rakenduse kood sõltub vähem andmebaasist, millega rakendus opereerib.



Pilt 1: Rakenduse kihid

Hibernate „tõlgib“ kõik päringud HQL keelde ja sealt edasi vajalikku SQL dialekti. Hibernate toetab dialekte nagu: DB2, PostgreSQL, MySQL, Oracle, Sybase, Microsoft SQL Server, SAP DB, Informix, Hypersonic SQL, H2 Database, Ingres, Progress, Mckoi SQL, Interbase, Pointbase, FrontBase ja Firebird.

Hibernate-i versioon on 2011. aasta oktoobri seisuga 4 ja seda arendab Jboss Community.

5.3 Mis on ORM?

Java on objekt-orienteeritud programmeerimiskeel ja seal kasutatavad objektid ei sobi nii mõneski mõttes kokku relatsiooniliste andmebaaside kahemõõtmelise mudeliga. Seda nimetatakse objekt-orienteeritud maailma ja relatsioonilise maailma paradigmade sobimatuseks ja sellest tulenevad paljud probleemid. Java võimaldab näiteks objektide samasust kontrollida järgmistel viisidel:

```
objekt1 == objekt2
```
Antud juhul tähendab koodijupi tagastatud tõene tõeväärtus seda, et tegu on täpselt sama objektiga, mis paikneb samas mälupiirkonnas.

```
objekt1.equals(objekt2)
```
See koodijupp kontrollib vaid objektide võrdsust, olenemata nende asukohast mälus.

Relatsioonilises tabelis aga eristatakse objekte ehk kirjeid vaid rea andmete järgi, ühes tabelis muudab objekti unikaalseks tema primaarvõti ja see on ainus kontroll objektide samasuse tuvastamiseks.

Java objektidega kaasneb pärilus, küsimused tekivad, kui püüame objektide omavaheliseid pärilussuhteid tabelitesse jäädvustada. Kas seda teha eraldi tabelite ja lisaidentifikaatorite abil? Kas suruda kogu pärilusahel ühte tabelisse? Ja kui raske on lõpuks kirjutada SQL-päringuid, et objekte saaks alati korrektselt küsida ja salvestada. Kui mitu korda oleks optimaalne kasutada päringus INNER JOIN käsku?

Lisaks eksisteerivad objektorienteeritud maailmas konstruktid, mida relatsioonilistes tabelites üldjuhul ei näe. Hea näide sellest on tänavaaadress. Objektorienteeritud koodis oleks loogiline hoida aadressi eraldi klassis, mitte seda igale aadressi omavale objektile väljana juurde kirjutada. Relatsioonilises andmebaasis aga lahendataks ülalkirjeldatud juhtum lihtsalt ühe tabeliveeruga, mis sisaldab aadresse.

ORM püüabki lahendada neid probleeme ehk tagada Java objektide automatiseeritud püsiesitus relatsiooniliste andmebaaside kaudu. (Bauer & King, 2005)

5.4 Millised on Hibernate raamistiku alternatiivid?

Puhast JDBC-d kasutades realiseeritakse püsiesitus käsitsi. Peale Hibernate-i on selleks loodud ka teisi raamistikke. Tavaliselt pakuvad sellised raamistikud peale objektide kaardistamise andmebaasidesse ka laiska laadimist (ingl. k. *lazy loading*) ja objektide puhverdamist. Puhverdamine võimaldab päringute arvu kokku hoida, sest korduvalt kasutatavaid objekte ei küsita andmebaasist kohe uuesti vaid need viibivad mõnda aega vahemälus, näiteks Hibernate kasutab hetktõmmiseid, et tuvastada, kas objekti on muudetud. Hibernate pakub kahetasandilist puhverdamist, millest üks on sisseehitatud ja alati töös, teine valikuline. Esimese tasandi puhver tagab, et rakendus tegeleb alati andmebaasiga kooskõlas oleva objektiga. See puhver sekkub ka siis, kui rakenduse erinevad lõimed püüavad teha muudatust samal tabelireal. Laisk laadimine võimaldab objektide instantseerimisega oodata kuni neid reaalselt pruugitakse. (Seddighi, 2009)

ORM raamistikud kasutavad tavaliselt mingit sorti kaardistamise definitsioonifaile, näiteks XML keeles. Nende abil saab raamistik aru, kuidas objekti ja tema püsiesitust tagavaid välju andmebaasi jäädvustada. Tavaliselt on sellisel raamistikul olemas Java funktsioonid andmebaasioperatsioonide tegemiseks.

5.4.1 Enterprise JavaBeans(EJB)

on Java EE (Enterprise Edition) jaoks mõeldud raamistik. Enamasti peetakse EJB-i keerukaks, mitteläbipaistvaks alternatiiviks. Lisaks on EJB serveripoolne komponent ja vajab seega konteinerit. (Seddighi, 2009)

5.4.2 Ibatis SQL Map

kasutab XML faile, et kaardistada päringute muutujaid andmebaasidesse ja vastupidi, sealt objektidesse. Töö käigus asendadakse muutujad SQL-stringide definitsioonides. Kahjuks ei paku raamistik lisaabstraktsioonikihti nagu Hibernate. (Seddighi, 2009)

5.4.3 Java Data Objects (JDO)

pakub samuti objektide püsiesitust, toetab erinevaid andmebaase, metaandmete kaudu objektide kaardistamist ja pakub ka oma päringukeelt, JDOQL-i. (Seddighi, 2009)

5.4.4 TopLink

on raamistik kasutamiseks nii JEE kui ka JSE keskkonnas ja pakub visuaalset kaardistamistööriista, et kaardistada suurt hulka erinevaid objekte ja seoseid. Toetatakse relatsioonilist kaardistamist, objektide kaardistamist XML failidesse ja JAXB-i. Raamistikul on abstraktsioonikiht püsiesitatavate objektidega tegelemiseks Java koodi tasandil. (Seddighi, 2009)

5.5 Hibernate-i omapära

Hibernate on neilt kõigilt häid ideid laenanud ja erineb oma eelkäijatest selle poolest, et on kergemini õpitav ja kasutatav, läbinähtavam ja ei vaja rakendusserverit. Hibernate on eeskujulikult dokumenteeritud ja raamistiku kasutamiseks on hulk raamatuid ja juhiseid internetis. Hibernate vajab töötamiseks vaid J2SE 1.2 või uuemat versiooni.

6 Hibernate olulisemad andmeobjektid

Hibernate-i konfiguratsioonifail on XML või properties vormingus ja hoiab andmebaasiga ühenduse loomiseks tarvilikku informatsiooni: kasutajanime, parooli, URL-i, draiveri klassi ja SQL-dialekti nimetust.

Oluline osa Hibernate raamistikust on püsiesitusvõimelised objektid, mis esinevad POJO-de kujul ehk konkreetse struktuuriga klassidena, millel puuduvad erifunktsioonid. Hibernate raamistiku tarvis loodud korrektsel POJO-l on tavaliselt väljad, getter ja setter meetodid, argumentideta konstruktor (kopeerimise vältimiseks). Lisaks tuleb üle kirjutada klassi hashCode() ja equals() meetodid. Viimased meetmed just paradigmade sobimatuse tõttu. HashCode() meetodit kasutavad näiteks hashMap, hashTable ja hashSet stuktuurid. Equals() on tarvis üle kirjutada, et objektide samasust oleks võimalik tuvastada. Ülekirjutused ei ole kohustuslikud.

Hibernate kaardistab objektid spetsiaalsete XML failide või annotatsioonide abil. Deklaratsioonides on alati olemas klass, mida kaardistatakse, tabel, kuhu väljad kirjutatakse, unikaalse ID äramärkimine ja seadistused, kuidas seda kõike tehakse.

Kõige lihtsamal juhul tuleb Hibernate-i kasutusele võtmiseks läbida 3 sammu:

1) Luua püsiesituse võimega klassid ehk rakenduses kasutatavad entiteedid.
2) Luua objektide jäädvustamiseks tabelid. Tavaliselt vastab üks tabel ühele klassile. Tabelite või objektide loomine ei pruugi olla kohustuslik, paljudel integreeritud programeerimiskeskkondadel on olemas utiliidid tabelitest objektide loomiseks ja vastupidi.
3) Luua kaardistamisfailid või annotatsioonid, mis kirjeldavad objektide jäädvustamist andmebaasi.

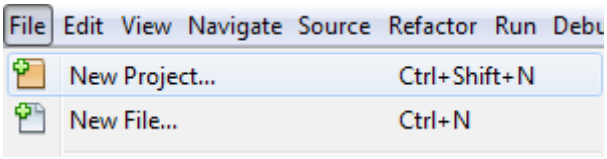
7 NetBeans IDE seadistamine

NetBeans IDE on saadaval järgnevalt aadressilt: http://netbeans.org/downloads/index.html
Hibernate-ga töötamiseks piisab Java SE versioonist.

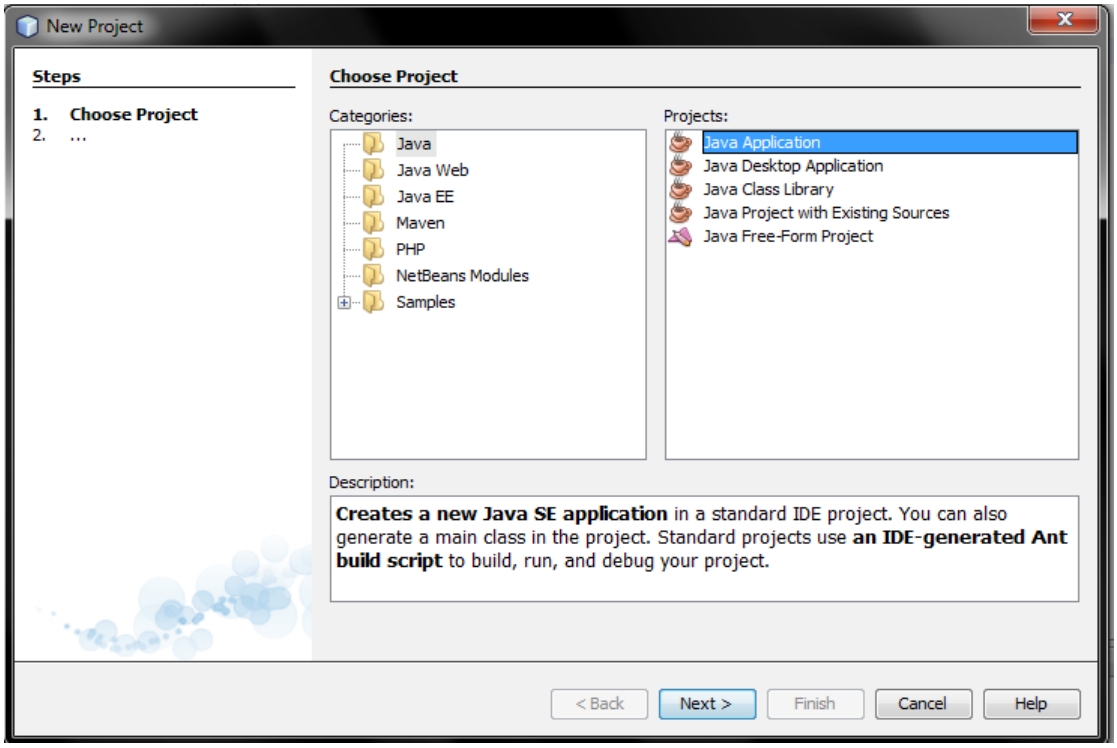
Hibernate-i failid võib vabalt lasta alla laadida Maven-il:
http://maven.apache.org/

Antud õppematerjali tegemisel laadis autor siiski vajalikud .jar failid alla käsitsi, mis on saadaval siit: http://sourceforge.net/projects/hibernate/files/hibernate3/

Kui vajalikud failid olemas ja NetBeans IDE töötab, võib luua uue Java Application tüüpi projekti:

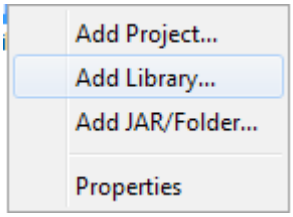


Pilt 2: Uue projekti loomine



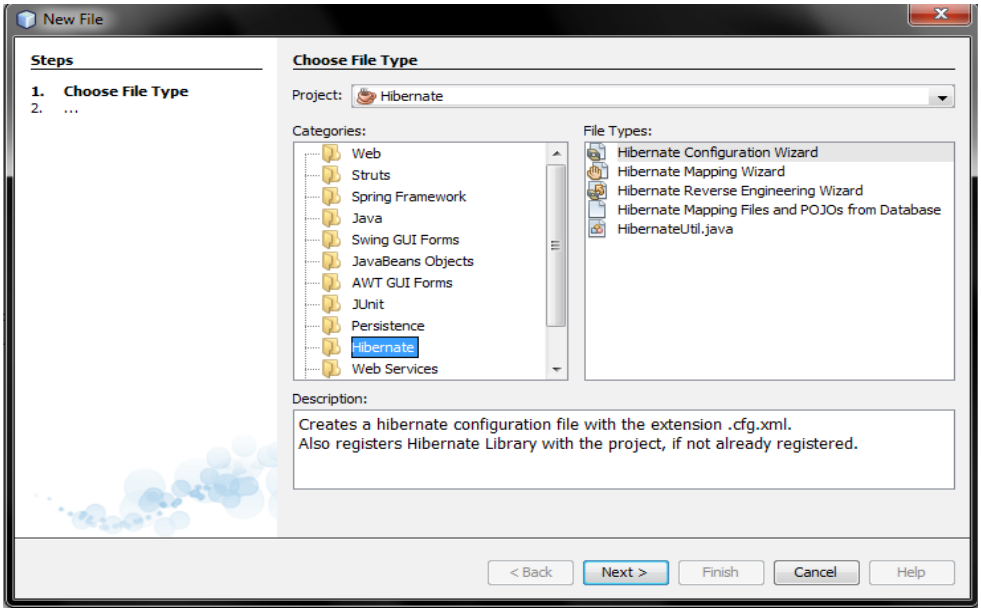
Pilt 3: Projektitüübi valik

Projektile tekitatakse automaatselt *Libraries* kaust, mille alla tuleks lisada Hibernate-i jar failid.



Pilt 4: Hibernate-i failide lisamine

Lihtsaim variant oleks paremklõpsu peale avanevast menüüst valida *Add Library...* ning valida sealt NetBeans-i sisseehitatud Hibernate JPA Library.



Pilt 5: Hibernate mallid

8 Andmebaasi paigaldamine

Kui antud õppematerjali kasutajal veel katsetamiseks sobilikku andmebaasi pole, saab selle vähese vaevaga paigaldada. Töö autor kasutas oma koodinäidetes MySQL Community Server 5.5.17 versiooni, mis on saadaval siit: http://www.mysql.com/downloads/mysql/

MySQL paigaldamine läbi kiirseadistuse kohaliku masina teenusena ei tohiks palju aega võtta. Töö autor seadistas oma andmebaasi töötama pordil 3306 ja määras juurkasutaja (*root*) parooliks '123'.

Kõik tööks vajalik peaks nüüd olemas olema.

Kui MySQL teenus ei ole määratud automaatselt käivituma, saab selle käivitada käsitsi. *Run* programmi tuleks sisestada 'services.msc' ja sealt paremklõpsuga MySQL käivitada. Et andmebaasile konsooli kaudu ligi pääseda, võib *Run* programmi sisestada 'cmd', liikuda *cd..* ja *dir* käskude abil MySQL *bin* kausta ja sisestada käsk 'mysql -ukasutajanimi -pparool', antud juhul 'mysql -uroot -p123'. Andmebaasi tabelitega saab tutvuda tavaliste MySQL käskude abil.

9 Esimene näide: „Hello World“

9.1 Püsiesitusega klassi loomine POJO mudeli järgi

Klass Isik realiseerib java.io.Serializable liidest, tal on väljad id, eesnimi, perekonnanimi. POJO mudeli kohaselt peavad objektil olema privaatsed väljad, avalikud meetodid nende väljade muutmiseks ja nende väärtuste küsimiseks ning tühi argumentideta konstruktor.

```
package Esimene;

public class Isik implements java.io.Serializable {
    private int id;
    private String eesnimi;
    private String perekonnanimi;
    public Isik() {

    }
    public Isik(int id, String eesnimi, String perekonnanimi) {
        this.id = id;
        this.eesnimi = eesnimi;
        this.perekonnanimi = perekonnanimi;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getEesnimi() {
        return eesnimi;
    }
    public void setEesnimi(String eesnimi) {
        this.eesnimi = eesnimi;
    }
    public String getPerekonnanimi() {
        return perekonnanimi;
    }
    public void setPerekonnanimi(String perekonnanimi) {
        this.perekonnanimi = perekonnanimi;
    }
}
```

Koodinäide 1: Klass Isik

NetBeans IDE võimaldab *getter* ja *setter* meetodid luua automaatselt *Refactor* menüüst läbi *Encapsulate Fields…* valiku.

Kuna Isik klassist objekti võidakse lisada hashMap, hashSet või hashTable tüüpi struktuuri, tuleb üle kirjutada funktsioon hashCode() ning kuna soovitakse ka Isik tüüpi objekte andmebaasi tabeli põhjal luua, tuleb kindlustada objekti identifitseerimine equals() funktsiooni ülekirjutamise kaudu. Ülekirjutamiseks kasutatakse Java keeles @Override annotatsiooni:

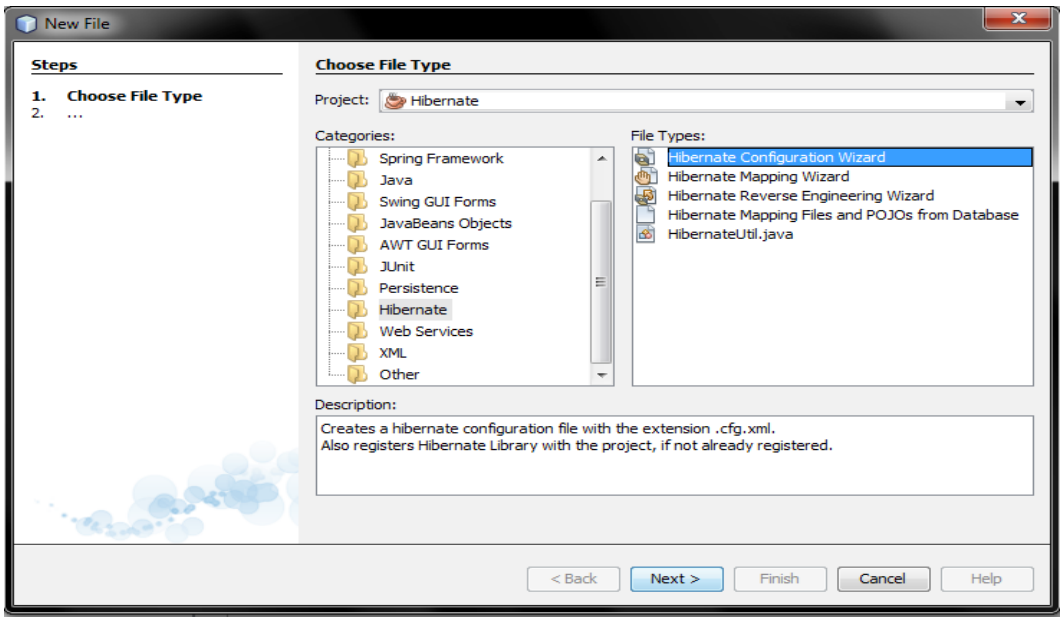
```
@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (!(o instanceof Isik)) {
        return false;
    }
    final Isik isik = (Isik) o;
    if (id == 0) {
        return false;
    }
    if (eesnimi != null ? !eesnimi.equals(isik.eesnimi) :
    isik.eesnimi != null) {
        return false;
    }
    if (perekonnanimi != null ? !perekonnanimi.equals(isik.perekonnanimi) :
    isik.perekonnanimi != null) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 53 * hash + this.id;
    hash = 53 * hash + (this.eesnimi != null ? this.eesnimi.hashCode() : 0);
    hash = 53 * hash + (this.perekonnanimi != null ? this.perekonnanimi.hashCode() : 0);
    return hash;
}
```

Koodinäide 2: equals() ja hashCode() meetodid

Need pealtnäha keerulised meetodid ei ole kohustuslikud rakenduse osad.

9.2 Hibernate konfiguratsioonifaili loomine

Et objekti relatsioonilisse andmebaasi üle kanda, tuleks esmalt teavitada Hibernate raamistikku andmebaasi asukohast, kasutajanimest ja paroolist, selleks kasutame *Hibernate Configuration Wizard*-nimelist viisardit.

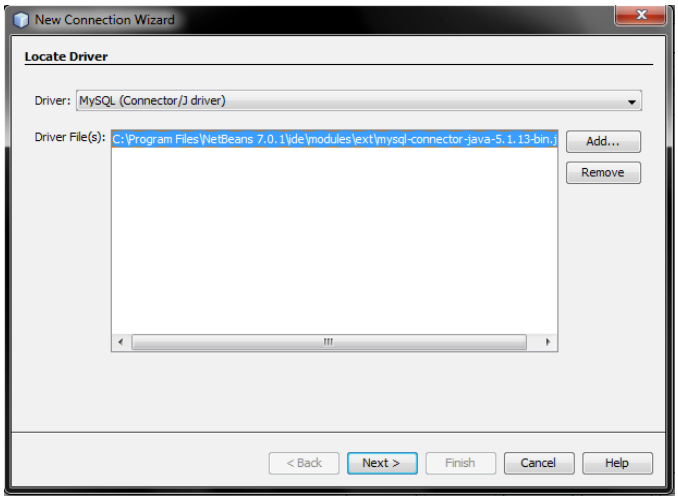


Pilt 6: Hibernate Configuration Wizard

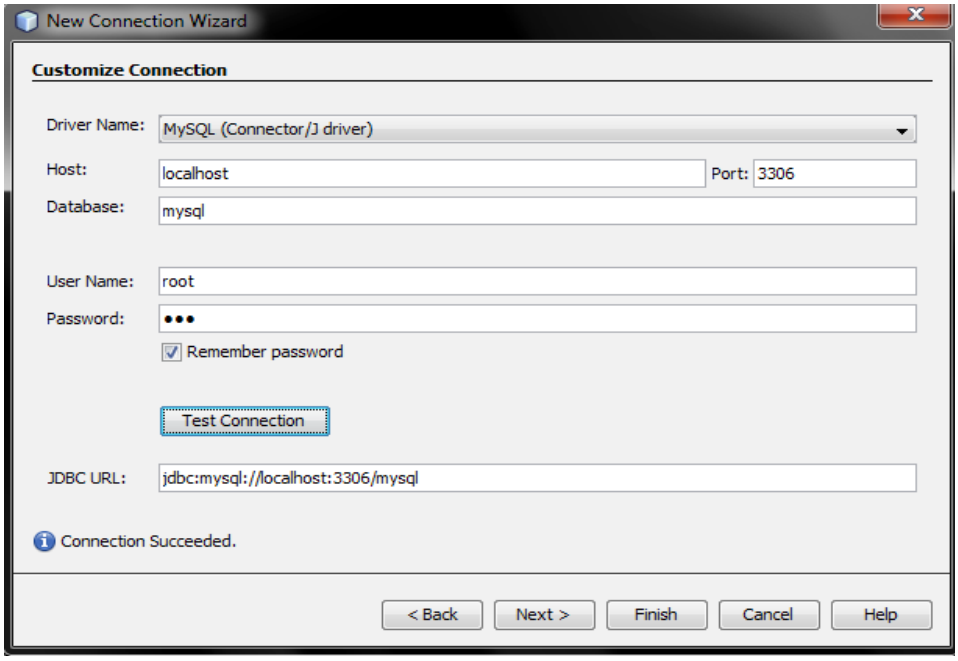
Miski ei keela ülaltoodud seadistusfaili loomist käsitsi, nõutud laiendid on .cfg.xml. On oluline, et arvutis oleks olemas JDBC draiver võimaldamaks suhtlust Java ja andmebaasi vahel. Näiteks MySQL JDBC draiver on saadaval siit: http://dev.mysql.com/downloads/connector/j/

JDBC draiveri paigaldamiseks käsitsi tuleb .jar fail lisada Libraries kausta.

Üldjuhul on levinumad JDBC draiverid NetBeans IDE-ga kaasas:



Pilt 7: JDBC Connectori valik



Pilt 8: Ühenduse seaded

Viisardi läbimise tulemuseks on alljärgnev Hibernate konfiguratsioonifail:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mysql</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">123</property>
  </session-factory>
</hibernate-configuration>
```

Koodinäide 3: Hibernate konfiguratsioonifail

Lisame seadistusfailile järgmised read:

```
<property name="hibernate.show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
```

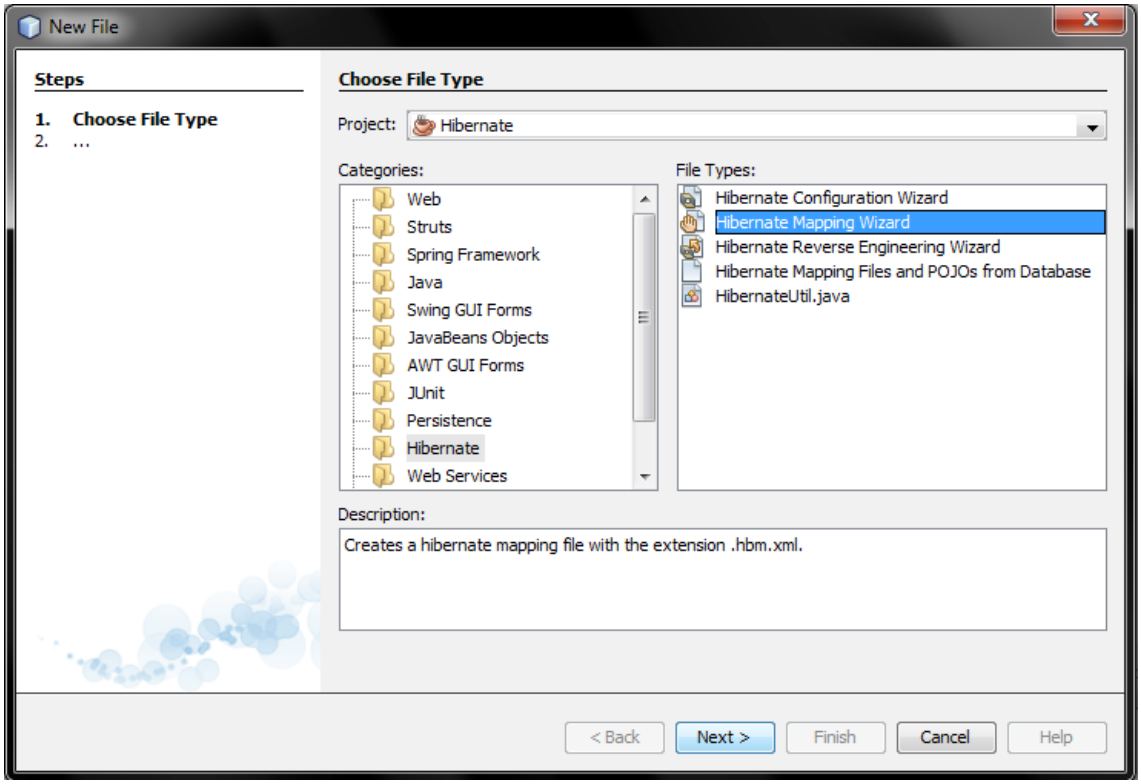
Esimene seadistus käsib Hibernate-l teha iga andmebaasioperatsiooni juures väljatrükk konsoolile ja teine annab loa raamistikul tabeleid luua.

Täielik seadistuste nimekiri on olemas NetBeans-i keskkonnas ja dokumentatsioon on saadaval siit: http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html

Tasub tähele panna, et seadistusfail on vaikimisi pakettideülene.

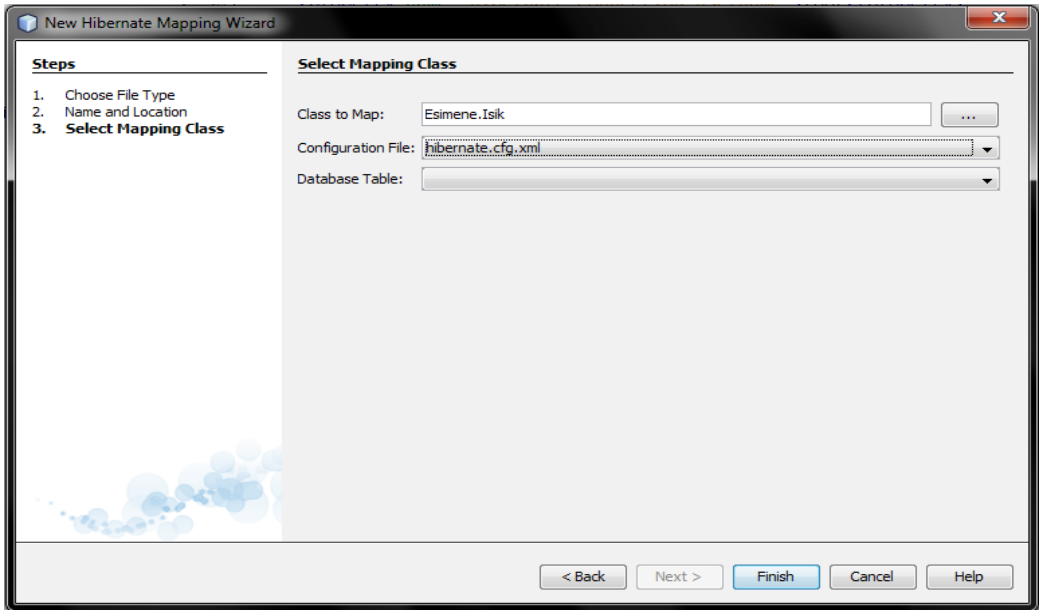
9.3 Objekti kaardistamine, Hibernate Mapping faili loomine

Mapping deklaratsioonifailis on kirjas, kuidas mingist klassist objekte andmebaasi talletada.



Pilt 9: Hibernate Mapping Wizard 1

Ka selle faili võib käsitsi luua, vajalikud laiendid on .hbm.xml.



Pilt 10: Hibernate Mapping Wizard 2

Nagu eelneval pildil näha, pole klass Isik paketist Esimene seotud ühegi kindla tabeliga. Kuna eelnevalt sai Hibernate seadistatud nii, et ta võib ise tabeleid luua, polegi programmeerijal vaja seda käsitsi teha.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class dynamic-insert="false" dynamic-update="false" mutable="true" name="Esimene.Isik"
optimistic-lock="version" polymorphism="implicit" select-before-update="false"></class>
</hibernate-mapping>
```

Koodinäide 4: Hibernate Mapping fail 1

Nagu näha, selle tulemusel, et Hibernate tabelist teadlik pole, ei genereerinud raamistik ka kaardistust. Mäletatavasti on Isik klassil 3 välja: id, eesnimi ja perekonnanimi. Hibernate peab teadma, kuidas neid andmebaasi talletada. Täiendatud kaardistusfail näeks seega välja selline:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class dynamic-insert="false" dynamic-update="false" mutable="true"
  name="Esimene.Isik" optimistic-lock="version" polymorphism="implicit"
  select-before-update="false">
      <id name="id" type="int">
        <column name="ID"></column>
        <generator class="increment"></generator>
      </id>
      <property name="eesnimi" type="string">
        <column name="EESNIMI" length="64"></column>
      </property>
      <property name="perekonnanimi" type="string">
        <column name="PEREKONNANIMI" length="64"></column>
      </property>
  </class>
</hibernate-mapping>
```

Koodinäide 5: Hibernate Mapping fail 2

Ülaltoodud koodiga sai määratud, et id on täisarv tüüpi, jäädvustatakse tulpa ID ja kasutatakse *increment* tüüpi generaatorit ehk suurendatakse iga uue kirje puhul primaarvõtit 1 võrra. Eesnimi ja perekonnanimi lähevad vastavatesse 64-märgilise pikkusega stringide tulpadesse.

Et raamistik oleks kaardistatud klassist teadlik, tuleb Hibernate seadistusfaili (.cfg.xml) lisada järgmine rida (juhul, kui NetBeans IDE seda juba ei teinud):

```
<mapping resource="Isik.hbm.xml"/>
```

9.4 Testsisestus

```
package Esimene;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Test1 {
    public static void main(String[] kapsas) {
        Configuration c = new Configuration().configure();
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try
        {
            Isik isik = new Isik();
            isik.setEesnimi("Ants");
            isik.setPerekonnanimi("Tamm");
            s.save(isik);
            tx.commit();
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 6: Esimese näite testprogramm 1

Ülaltoodud programm sisestab loodud isik objekti andmebaasi, kusjuures id genereeritakse automaatselt. Tabeli nimeks saab isik, kuna muud nime ei täpsustatud. Tulbad on ID, EESNIMI, PEREKONNANIMI, nii nagu kaardistusfailid nõutud. Täiendades aga programmi järgnevate ridadega:

```
Isik isik2 = new Isik();
isik2.setEesnimi("Sviit");
isik2.setPerekonnanimi("Hõum");
s.save(isik2);
```

Tekib andmebaasi järgmine kirje isikust Sviit Hõum, kelle id järjekorras järgmine number, 2.

Objekti andmebaasist kustutamiseks:

```
s.delete(isik2);
```

Nagu näha, on võimalik muudatusi kas sisse kanda või tagasi võtta *Transaction* objekti kaudu *commit()* või *rollback()* funktsioone kasutades.

Configuration parameetriteta funktsioon *configure()* eeldab, et seadistusfail asub projekti juurkaustas. Kui see asub mujal, tuleb faili suhteline asukoht stringi kujul ette anda.

Väljavõte andmebaasist:

Pilt 11: Tabel isik

9.5 Andmebaasist objekti küsimine

Eelmises näites lisasime andmebaasi 2 kirjet. Järgmise koodijupi ülesandeks on kirjed taas objektideks muuta. Et Hibernate tabelit teise testprogrammi jooksutamisel ei kustutaks ja uuesti looks, muudame ära .cfg.xml faili. Vana:

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

Uus:

```
<property name="hibernate.hbm2ddl.auto">update</property>
```

```
package Esimene;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Test2 {
    public static void main(String[] kapsas) {
        Configuration c = new Configuration().configure();
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        try
        {
            Isik isik = (Isik)s.get(Isik.class, 1);
            System.out.println(isik.getEesnimi());
            Isik isik2 = (Isik)s.get(Isik.class, 2);
            System.out.println(isik2.getEesnimi());
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
        }
        s.close();
    }
}
```

Koodinäide 7: Hibernate objekti küsimine

Andmebaasist objekti küsimiseks ei ole tarvis kasutada Transaction objekti, küll aga sinna kirjutamiseks. Tasub tähele panna, et andmebaasist tulev info tuleb *castida* õigest klassist objektiks, et pääseda ligi tema objektorienteeritud kujule.

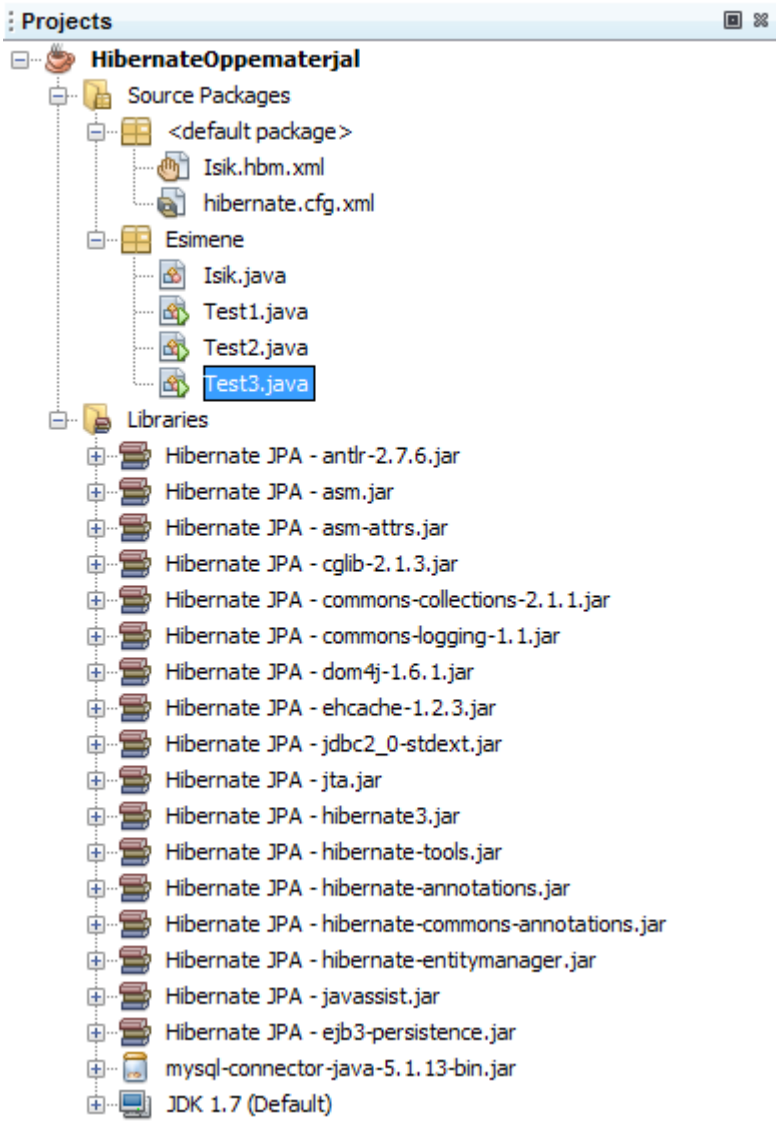
9.6 Objektid päringu järgi

Hibernate raamistik pakub selleks oma Criteria API-t.

```
package Esimene;

import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Restrictions;

public class Test3 {
    public static void main(String[] kapsas) {
        Configuration c = new Configuration().configure();
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        try {
            Criteria cr = s.createCriteria(Isik.class);
            cr.add(Restrictions.between("id", 0, 1));
            List vastus = cr.list();
            for (int i = 0; i < vastus.size(); i++) {
                Isik isik = (Isik)vastus.get(i);
                System.out.println(isik.getEesnimi() + " " +
                  isik.getPerekonnanimi());
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
        s.close();
    }
}
```

Koodinäide 8: Hibernate Criteria

Criteria objekt loob Isik klassi põhjal võimalikud kriteeriumid objektide küsimiseks ja lisades sinna piirangu, mis tähistab, et vastusesse jäävad vaid kirjed id-ga nullist üheni kaasaarvatud, saamegi massiivi sobivate objektidega. Antud näites on kasutatud märksõna *between*, võimalikud on ka näiteks *eq*, *lt*, *le*, *gt*, *ge*, *like*, *isNull*, *isNotNull* jne. Dokumentatsioon: : http://docs.jboss.org/hibernate/core/3.3/reference/en/html/querycriteria.html



Pilt 12: Projektipuu

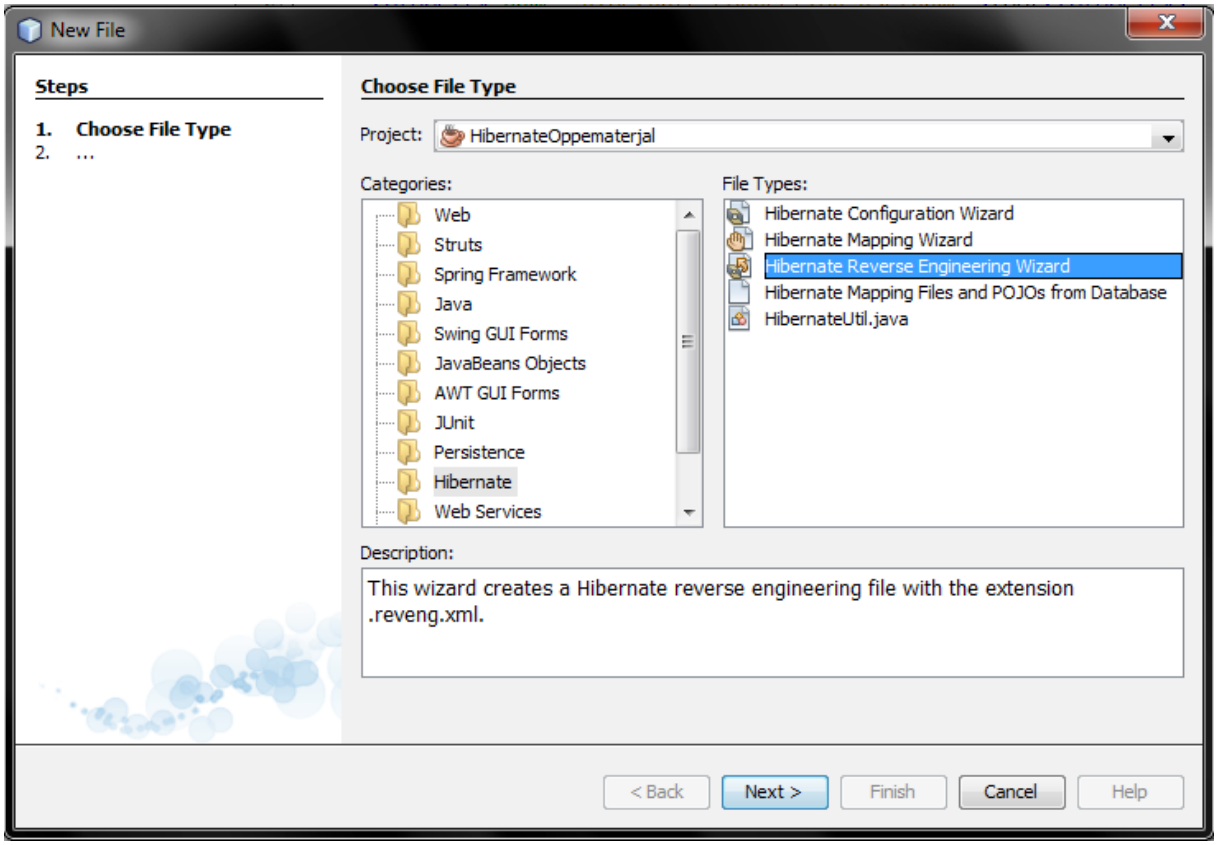
9.7 Hibernate Reverse Engineering Wizard

Eelnevas näites lõime POJO mudeli järgi klassi, kaardistusfaili klassi talletamiseks ja testprogrammi. NetBeans IDE pakub selle ülesande lahendamiseks ka muid tööriistu. Vaatleme veel mõnda viisardit.

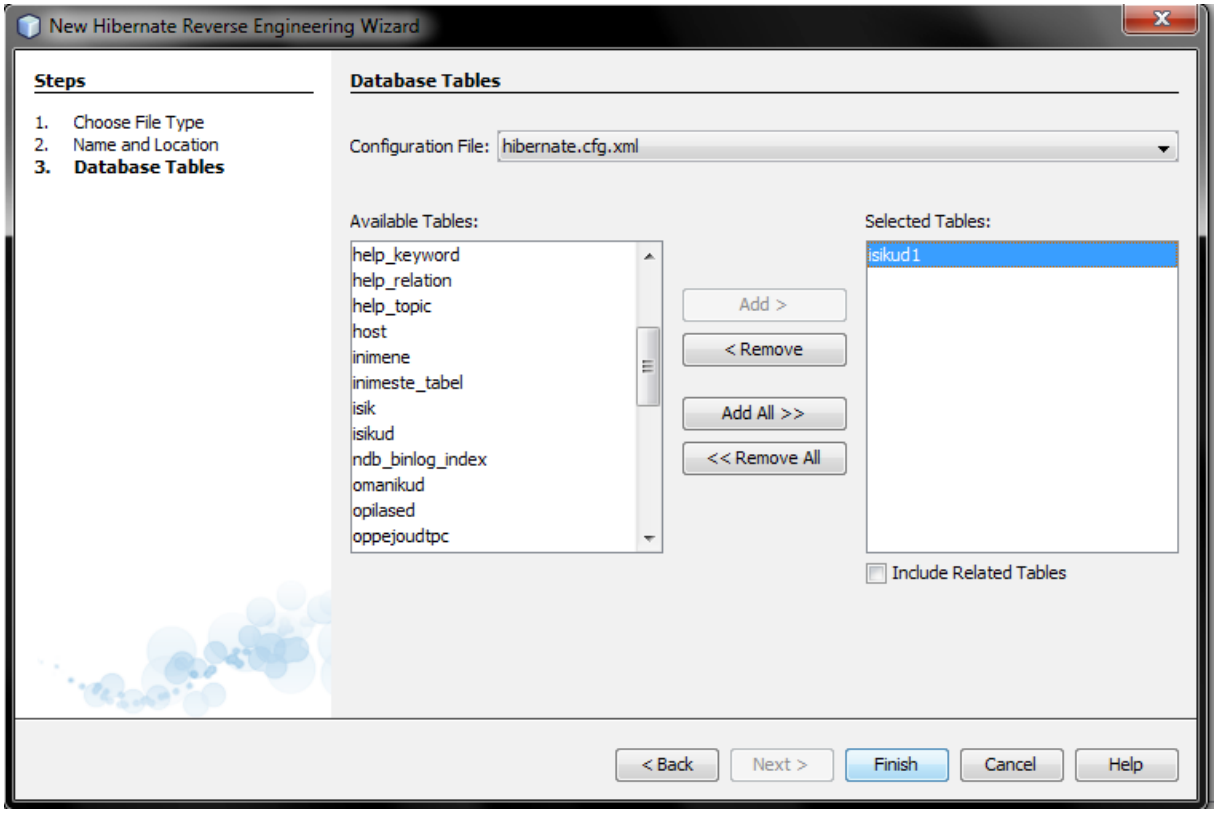
Seekord ei alga meie testrakenduse arendus mitte Java klassiga vaid andmebaasi tabeliga. Autor kasutas tabeli loomiseks MySQL konsooli ja käsku:

```
CREATE TABLE Isikud1 (id int not null auto_increment primary
key, eesnimi varchar(256), perekonnanimi varchar(256));
```

Kui tabel loodud, käivitame järgmise viisardi:



Pilt 13: Hibernate Reverse Engineering Wizard 1

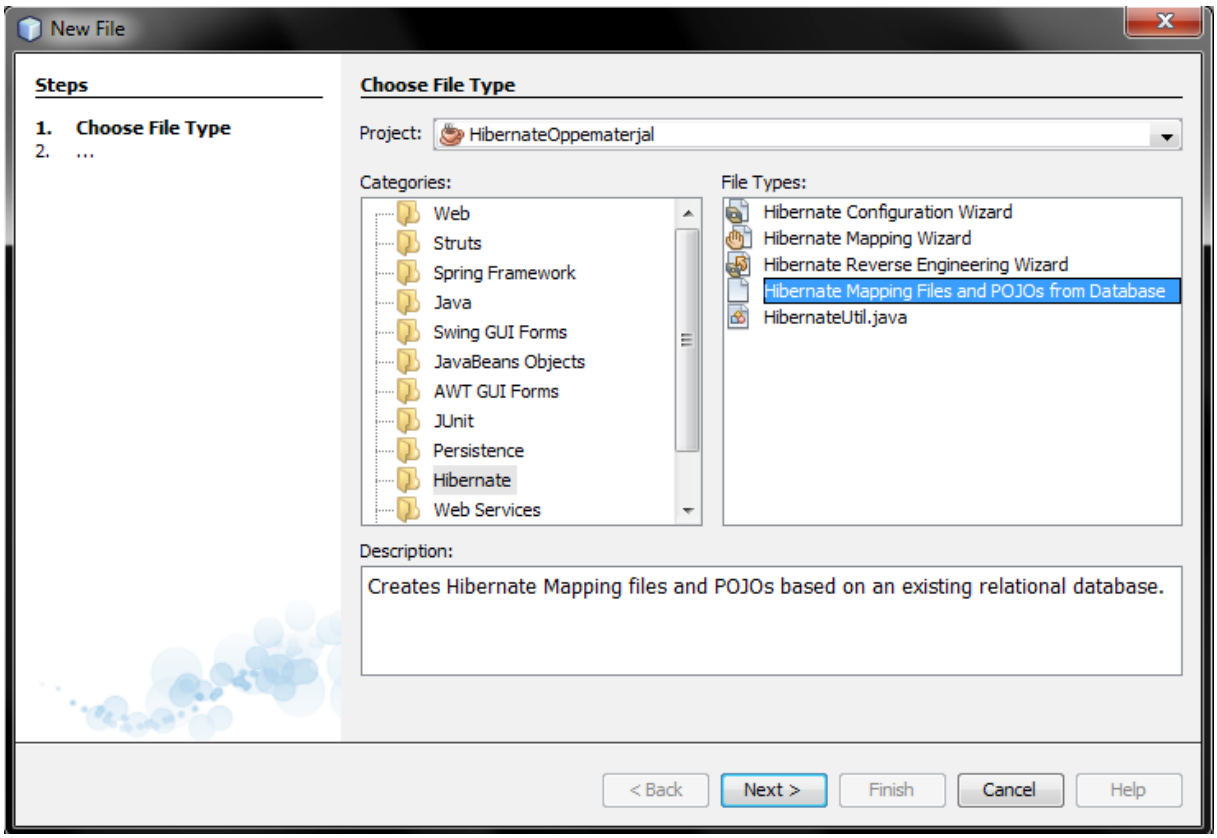


Pilt 14: Hibernate Reverse Engineering Wizard 2

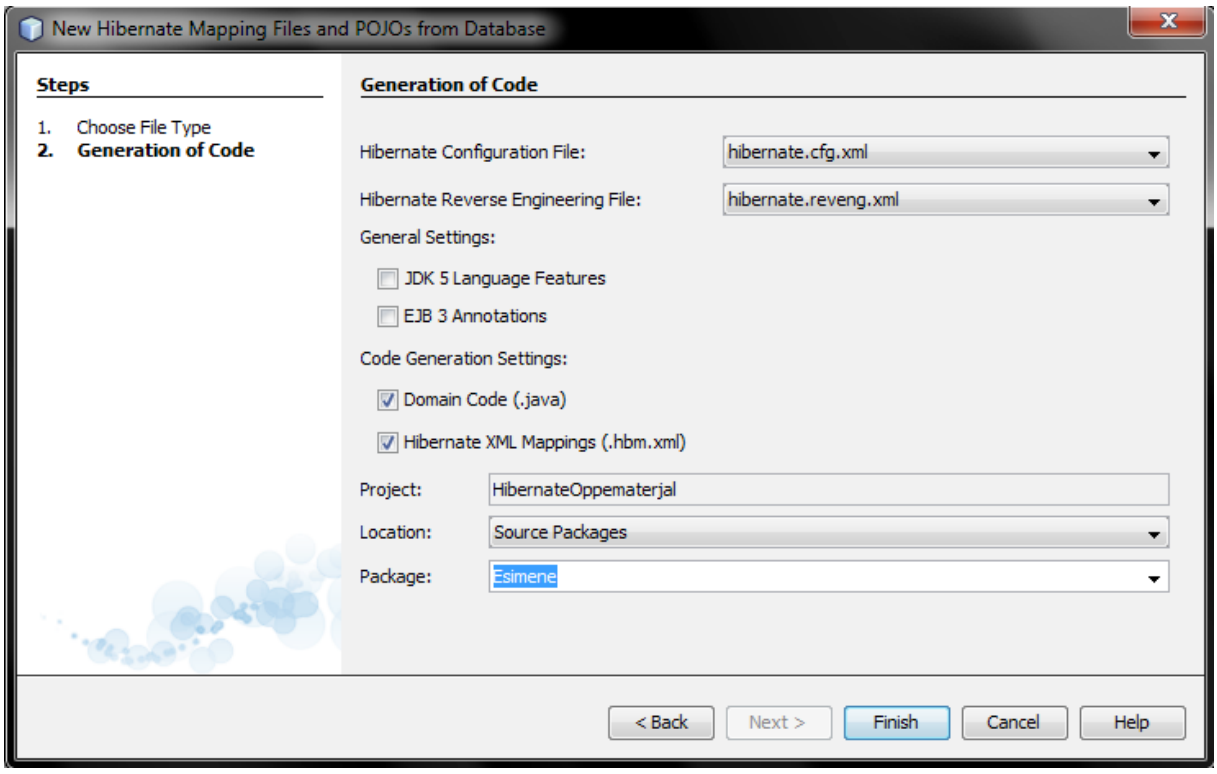
Genereeritakse järgmine .reveng.xml laiendiga fail:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse
Engineering DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-reverse-
engineering-3.0.dtd">
<hibernate-reverse-engineering>
  <schema-selection match-catalog="mysql"/>
  <table-filter match-name="isikud1"/>
</hibernate-reverse-engineering>
```

Koodinäide 9: Hibernate .reveng.xml fail



Pilt 15: Hibernate Mapping Files and POJOs from Database viisard 1



Pilt 16: Hibernate Mapping Files and POJOs from Database viisard 2

Viisardi läbimise tulemuseks on failid Isikud1.java ja Isikud1.hbm.xml. Loome programmi testimaks genereeritud faile:

```
package Esimene;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Test4 {
    public static void main(String[] kapsas) {
        Configuration c = new Configuration().configure();
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try
        {
            Isikud1 isik = new Isikud1();
            isik.setEesnimi("Ants");
            isik.setPerekonnanimi("Tamm");
            Isikud1 isik2 = new Isikud1();
            isik2.setEesnimi("Sviit");
            isik2.setPerekonnanimi("Hõum");
            s.save(isik);
            s.save(isik2);
            tx.commit();
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 10: Genereeritud failide testprogramm

Kood jäi üldjoontes samaks, mis Test1 programmis, asendasime vaid klassi Isik klassiga Isikud1 klassiga. NetBeans IDE kasutajad eelistavad tõenäoliselt viisardite meetodit käsitsi kirjutamisele.

9.8 Ülesanne

1) Looge POJO mudeli põhjal oma klass, millel välju muudest tüüpidest, kui int ja String.
2) Looge andmebaasi tabel, mis selliseid objekte talletab.
3) Kasutades Hibernate Mapping Wizard viisardit, ühendage objekt tabeliga.
4) Salvestage tabelisse 100 kirjet ja küsige Criteria objekti kaudu välja viimased 50.

10 Teine näide: annotatsioonid

Objektide kaardistamiseks on mitu võimalust, eelnevalt vaatlesime .hbm.xml kaardistusfaile, selles näites kasutame sama funktsionaalsuse realiseerimiseks annotatsioone javax.persistence kogust. Loome uue paketi, nimeks Teine.

```
package Teine;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="ISIKUD")
public class Isik implements java.io.Serializable {
    private int id;
    private String eesnimi;
    private String perekonnanimi;
    public Isik() {

    }
    public Isik(int id, String eesnimi, String perekonnanimi) {
        this.id = id;
        this.eesnimi = eesnimi;
        this.perekonnanimi = perekonnanimi;
    }
    @Id
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Basic
    @Column(name="EESNIMI")
    public String getEesnimi() {
        return eesnimi;
    }
    public void setEesnimi(String eesnimi) {
        this.eesnimi = eesnimi;
    }
    @Basic
    @Column(name="PEREKONNANIMI")
    public String getPerekonnanimi() {
        return perekonnanimi;
    }
    public void setPerekonnanimi(String perekonnanimi) {
        this.perekonnanimi = perekonnanimi;
    }
}
```

Koodinäide 11: Hibernate annotatsioonid

Kuna need antud näite puhul olulised ei ole, on välja jäetud hashCode() ja equals() ülekirjutused.

Loome uue paketi jaoks ka uue .cfg.xml faili, seekord paketi Teine sisse. Hibernate Configuration Wizard peaks olema juba tuttav tööriist. Mõistagi lisame read:

```
<property name="hibernate.hbm2ddl.auto">create</property>
<property name="hibernate.show_sql">true</property>
<mapping class="Teine.Isik"></mapping>
```

Annoteeritud klassi testimiseks sobib järgmine kood:

```
package Teine;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

public class Test1 {
    public static void main(String[] args) {
        SessionFactory sf = new
AnnotationConfiguration().configure("Teine/hibernate.cfg.xml").buildSessionFacto
ry();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try {
            Isik i = new Isik(1, "Reinu", "Vader");
            s.save(i);
            tx.commit();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 12: Hibernate annotatsioonide test

Nagu näha, tuli meetodile *configure* anda argumendina kaasa konfiguratsioonifaili asukoht.

```
mysql> select * from isikud;
+----+---------+---------------+
| ID | EESNIMI | PEREKONNANIMI |
+----+---------+---------------+
|  1 | Reinu   | Vader         |
+----+---------+---------------+
1 row in set (0.00 sec)

mysql>
```

Pilt 17: Annoteeritud klassi 'instants' andmebaasis

10.1 Ülesanne

1) Kasutage annotatsioone, et kaardistada enda väljamõeldud klass.
2) Looge testprogramm, mis sisestab andmebaasi mõned objektid.
3) Lisage testprogrammile funktsionaalsus, mis võimaldab objekte küsida.

11 Kolmas näide: pärilus

Hibernate pakub päriluse jäädvustamiseks kolme strateegiat:

1) Üks tabel iga konkreetse klassi kohta.
2) Üks tabel kõigi klasside kohta.
3) Üks tabel iga pärilusahela kohta.

Ka pärilussuhteid on võimalik kaardistada nii XML deklaratsioonifailide kui ka annotatsioonide kaudu.

11.1 Table Per Class

Esmalt vaatleme, kuidas realiseerida esimest strateegiat, inglise keeles *Table Per Class,* annotatsioonide kaudu. Abstraktne klass Inimene:

```
package Kolmas;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Inimene implements java.io.Serializable {
    private int id;
    private String nimi;
    public Inimene() {

    }
    public Inimene(int id, String nimi) {
        this.id = id;
        this.nimi = nimi;
    }
    @Id
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Basic
    @Column(name="NIMI")
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
}
```

Koodinäide 13: Table Per Class 1

Klassid Yliopilane ja Oppejoud pärinevad klassist Inimene:

```
package Kolmas;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("YLIOPILANE")
public class Yliopilane extends Inimene implements java.io.Serializable {
    private String matriklinumber;
    public String getMatriklinumber() {
        return matriklinumber;
    }
    public void setMatriklinumber(String matriklinumber) {
        this.matriklinumber = matriklinumber;
    }
}
```

Koodinäide 14: Klass Yliopilane

```
package Kolmas;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("OPPEJOUD")
public class Oppejoud extends Inimene implements java.io.Serializable {
    private String eriala;
    public String getEriala() {
        return eriala;
    }
    public void setEriala(String eriala) {
        this.eriala = eriala;
    }
}
```

Koodinäide 15: Klass Oppejoud

Loome ka kolmanda paketi tarvis uue konfiguratsioonifaili.

Nagu ikka, tuleb Hibernate seadistusfaili lisada seadistused vajalikud klassid:

```
<property name="hibernate.show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="Kolmas.Yliopilane"/>
<mapping class="Kolmas.Oppejoud"/>
```

Järgmisena loome testprogrammi.

```
package Kolmas;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

public class Test1 {
    public static void main(String[] args) {
        SessionFactory sf = new
AnnotationConfiguration().configure("Kolmas/hibernate.cfg.xml").buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try {
            Yliopilane yliopilane = new Yliopilane();
            yliopilane.setId(1);
            yliopilane.setNimi("Eksmati");
            yliopilane.setMatriklinumber("092472");
            Oppejoud oppejoud = new Oppejoud();
            oppejoud.setId(2);
            oppejoud.setNimi("Doktor Ahven");
            oppejoud.setEriala("Ihtüoloogia");
            s.save(yliopilane);
            s.save(oppejoud);
            tx.commit();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 16: Table Per Class testprogramm

Testfaili käivitamisel näeme, et andmebaasi tekivad tõepoolest tabelid YLIOPILANETPC ja OPPEJOUDTPC, kus salvestatud kõik objekti väljad, ka need, mis kuuluvad klassile Inimene. Abstraktse klassi jaoks tabelit pole.

```
mysql> select * from YLIOPILASEDTPC;
+----+---------+----------------+
| ID | NIMI    | matriklinumber |
+----+---------+----------------+
|  1 | Eksmati | 092472         |
+----+---------+----------------+
1 row in set (0.00 sec)

mysql> _
```

Pilt 18: Tabel YLIOPILASEDTPC

```
mysql> select * from OPPEJOUDTPC;
+----+--------------+--------------+
| ID | NIMI         | eriala       |
+----+--------------+--------------+
|  2 | Doktor Ahven | Iht╙oloogia  |
+----+--------------+--------------+
1 row in set (0.00 sec)
```

Pilt 19: Tabel OPPEJOUDTPC

Strateegia on ressursinõudlik ja ei ole hea valik suurte projektide jaoks.

11.2 Single Table

Antud strateegia talletab kogu klassihierarhia ühte tabelisse, kasutades spetsiaalset tulpa, et objektidel vahet teha. Vaatame, kuidas realiseerida *Single Table* lahendus annotatsioonide kaudu. Jällegi, XML deklaratsioonifailid võimaldavad sama.

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="INIMESETYYP",
discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("LIHTINIMENE")
public class Inimene implements java.io.Serializable {
...
}
```

Et tabelisse saaks tekitada ka LIHTINIMENE tüüpi kirjeid, muudame klassitüübi abstraktsest tavaklassiks. Nagu näha, on TABLE_PER_CLASS asendatud strateegiaga SINGLE_TABLE ja diskrimineeriva tulba nimeks määratud INIMESETYYP, mille väärtus antud klassi puhul oleks LIHTINIMENE.

Mõistagi tuleb lisada importkäsud:

```
import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
```

Samalaadsed muudatused tuleks sisse viia ka Yliopilane ja Oppejoud klassidesse:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("OPPEJOUD")
public class Oppejoud extends Inimene implements
java.io.Serializable {
...
}
```

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("YLIOPILANE")
public class Yliopilane extends Inimene implements
java.io.Serializable {
...
}
```

Importkäsud:

```
import javax.persistence.DiscriminatorValue;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
```

Testprogramm jääb suures osas samaks, lisame vaid objekti Inimene klassist:

```
Inimene inimene = new Inimene();
inimene.setId(3);
inimene.setNimi("Ants Tamm");
s.save(inimene);
```

Nagu andmebaasikirjetest näha võime, on tekkinud üks tabel kolmest erinevast klassist objektidega, mille tüüp määratud eraldi tulpa INIMESETYYP. Mõistagi peavad mõned kirje osad olema nullitavad, näiteks peab Oppejoud klassist objekti matriklinumber olema NULL, nagu ka Yliopilane klassist objekti eriala NULL. Tõsi, eriala ei pruukinud olla parim valik õppejou ja üliõpilase eristamiseks, sest ka üliõpilase õpitava eriala kohta saab öelda, et see on selle üliõpilase eriala. Single table strateegia on lihtne ja päringud ei vaja kunagi näiteks JOIN käske, küll aga võib keerulise klassihierarhia puhul tulla tabelisse liigselt NULL väärtusega lahtreid.

```
mysql> select * from inimene;
+-------------+----+-------------+-----------------+--------------+
| INIMESETYYP | ID | NIMI        | matriklinumber  | eriala       |
+-------------+----+-------------+-----------------+--------------+
| YLIOPILANE  |  1 | Eksmati     | 092472          | NULL         |
| OPPEJOUD    |  2 | Doktor Ahven | NULL           | Iht?oloogia  |
| LIHTINIMENE |  3 | Ants Tamm   | NULL            | NULL         |
+-------------+----+-------------+-----------------+--------------+
3 rows in set (0.00 sec)
```

Pilt 20: Tabel inimene (SINGLE_TABLE)

11.3 Joined, table per subclass

Kolmas strateegia jäädvustab erinevatest klassidest objektid küll eri tabelitesse, kuid alamklassi tabelis on tulbad vaid väljade kohta, mida ülemklassil pole. Vaatame, kuidas antud strateegiat realiseerida XML deklaratsioonifaili abil. Tekitame paketti Kolmas faili Joined.hbm.xml, mille sisuks:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="Kolmas.Inimene2" table="INIMESTE_TABEL">
        <id name="id" column="ID" type="int">
            <generator class="native"/>
        </id>
        <property name="nimi" column="NIMI" type="string"/>
        <joined-subclass name="Kolmas.Yliopilane2" table="YLIOPILASTE_TABEL">
            <key column="YLIOPILASTE_TABEL_ID"/>
            <property name="matriklinumber" column="MATRIKLINUMBER"
type="string"/>
        </joined-subclass>
        <joined-subclass name="Kolmas.Oppejoud2" table="OPPEJOUDUDE_TABEL">
            <key column="OPPEJOUDUDE_TABEL_ID"/>
            <property name="eriala" column="ERIALA" type="string"/>
        </joined-subclass>
    </class>
</hibernate-mapping>
```

Koodinäide 17: Table per subclass kaardistusfail

Kuna nüüd on kõik raamistikule vajalik kirjas .hbm.xml failis, ei ole annotatsioone ega importkäske tarvis. Autor lõi projektifailide säilitamise huvides uued klassid Inimene2, Oppejoud2 ja Yliopilane2.

```
public class Inimene2 implements java.io.Serializable {
...
}
```

Lisame paketi Kolmas .cfg.xml faili järgmise rea:

```
<mapping resource="Kolmas/Joined.hbm.xml"/>
```

Ja kommenteerime välja annotatsioonidega lahenduse kohta käivad read:

```
<!--<mapping class="Kolmas.Yliopilane"/>-->
<!--<mapping class="Kolmas.Oppejoud"/>-->
```

Uues testprogrammis asendame *AnnotationConfiguration* objekti *Configuration* objektiga:

```
SessionFactory sf = new
Configuration().configure('Kolmas/hibernate.cfg.xml').buildSes
sionFactory();
```

Testfaili käivitamisel näeme, et andmebaasi tekivad järgmised tabelid: inimeste_tabel, oppejoudude_tabel, yliopilaste_tabel. Nagu t*able per subclass* strateegia nõuab, sisaldab inimeste_tabel vaid id-d ja nime, oppejoudude_tabel on saanud primaarvõtme, mis viitab inimeste_tabel tabelile, lisaks on seal kirjas õppejõu eriala. Analoogselt on talitatud yliopilaste_tabel tabeli puhul.

Antud strateegia vajakajäämised hakkavad silma torkama, kui klasside hierarhia kasvab vertikaalselt. B on A järglane, C on B järglane jne. Et Hibernate kirjetest objekti looks, peab ta tegema päringu kogu klassihierarhia kohta, mis mõjub negatiivselt jõudlusele. Teiseks ei ole tabelile peale vaadates võimalik aru saada, kui kaugele klassihierarhia ulatub. (Seddighi, 2009)

Näiteks MySQL konsooli kaudu tabelit oppejoudude_tabel vaadates näeme vaid kirjeid unikaalsete identifikaatoritega ja eriala väljadega. Kuigi pealkirjast võib midagi välja lugeda, et ole meil võimalik isegi öelda, kas tegu ei ole mitte lihtsalt erialade tabeliga. Lisaks pole meil aimugi, kas rakenduses kasutatav objekt pärineb ühest ülemklassist või on ahel pikem.

```
mysql> select * from oppejoudude_tabel;
+---------------------+--------------+
| OPPEJOUDUDE_TABEL_ID | ERIALA       |
+---------------------+--------------+
|                   2 | Iht╨oloogia  |
+---------------------+--------------+
1 row in set (0.00 sec)

mysql> select * from yliopilaste_tabel;
+---------------------+----------------+
| YLIOPILASTE_TABEL_ID | MATRIKLINUMBER |
+---------------------+----------------+
|                   1 | 092472         |
+---------------------+----------------+
1 row in set (0.00 sec)

mysql> select * from inimeste_tabel;
+----+--------------+
| ID | NIMI         |
+----+--------------+
|  1 | Eksmati      |
|  2 | Doktor Ahven |
|  3 | Ants Tamm    |
+----+--------------+
3 rows in set (0.00 sec)
```

Pilt 21: Table per subclass strateegia abil tekitatud tabelid

11.4 Ülesanne

1) Looge klassihierarhia, milles vähemalt 3 astet. Näiteks sõiduk-veesõiduk-kalapaat.
2) Valige enda arvates sobivaim strateegia antud klasside andmebaasi talletamiseks.
3) Looge testprogramm, milles kutsute välja klassihierarhia ülemisest klassist objekti meetodi alumisest klassist objekti instantsilt. Näiteks on sõidukil üldine meetod getNimi(), mille kutsute välja ühe kalapaat klassi instantsi kaudu.
4) Püüdke realiseerida samad klassid kahe ülejäänud strateegiaga.

12 Komponentide kaardistamine

Hibernate raamistikus kasutatakse komponente olukorras, kus objektorienteeritud maailmas on loogiline luua uus klass millegi jaoks, mis relatsioonilises maailmas talletataks samasse tabelisse. Hea näide on aadress. Kui Omanik klassil on väli, mis viitab mingile Aadress objektile ja seda mujal ei kasutata, tuleb appi komponentide kaardistamine. Komponentide kasutamise peamine põhjus on, et see teeb rakenduse paremini mõistetavaks. Oluline on mõista, et komponendid ei eksisteeri rakenduses kunagi iseseisvalt. Vaatleme, kuidas realiseerida komponenti annotatsioonide kaudu:

```
package Neljas;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="OMANIKUD")
public class Omanik implements java.io.Serializable {
    @Id
    @Column(name="ID")
    private int id;
    @Basic
    @Column(name="NIMI")
    private String nimi;
    @Embedded
    private Aadress aadress;
    public Omanik() {

    }
    public Omanik(int id, String nimi, Aadress aadress) {
        this.id = id;
        this.nimi = nimi;
        this.aadress = aadress;
    }
        public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
        public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    public Aadress getAadress() {
        return aadress;
    }
    public void setAadress(Aadress aadress) {
        this.aadress = aadress;
    }
}
```

Koodinäide 18: Klass Omanik

Mõistagi on tarvis klassid Omanik ja Aadress .cfg.xml failis deklareerida.

```
package Neljas;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class Aadress implements java.io.Serializable {
    @Column(name="AADRESS")
    private String adre;
    public String getAdre() {
        return adre;
    }
    public void setAdre(String adre) {
        this.adre = adre;
    }
}
```

Koodinäide 19: Klass Aadress

```
package Neljas;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

public class Test1 {
    public static void main(String[] kapsas) {
        AnnotationConfiguration c = new
AnnotationConfiguration().configure("Neljas/hibernate.cfg.xml");
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try
        {
            Omanik omanik = new Omanik();
            Aadress omanikuaadress = new Aadress();
            omanikuaadress.setAdre("Seene 1");
            omanik.setId(1);
            omanik.setNimi("Myko");
            omanik.setAadress(omanikuaadress);
            s.save(omanik);
            tx.commit();
            Omanik andmebaasist = (Omanik)s.get(Omanik.class, 1);
            System.out.println(andmebaasist.getAadress().getAdre());
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 20: Komponendi testprogramm

Nagu näha, talletas Hibernate aadressi küll samasse tabelisse, mis omaniku, kuid objektorienteeritud rakenduses eksisteerib aadress omanikuga seotud objektina.

```
mysql> select * from omanikud;
+----+---------+------+
| ID | AADRESS | NIMI |
+----+---------+------+
|  1 | Seene 1 | Myko |
+----+---------+------+
1 row in set (0.04 sec)
```

Pilt 22: Tabel OMANIKUD

12.1 Ülesanne

1) Realiseerida mingi uus komponendiga klass, näiteks võib kirjutada isikule komponendi nimi, mis koosneb eesnimest ja perekonnanimest.
2) Looge testprogramm.

13 Andmemassiivide kaardistamine

Tihtilugu on Java objektidega seotud mingit sorti massiivid. Hibernate võimaldab andmebaasi talletada Java *Collection*, *List*, *Set*, *SortedSet*, *Map* ja *SortedMap* massiive. Järgmises näites talletame Autor klassiga seotud teoste nimekirja, mis antud juhul on *List* tüüpi andmemassiiv. Kaardistamiseks kasutame deklaratsioonifaili.

```
package Viies;

import java.util.List;

public class Autor implements java.io.Serializable {
    private int id;
    private String nimi;
    private List teosed;
    public Autor() {

    }
    public Autor(int id, String nimi, List teosed) {
        this.id = id;
        this.nimi = nimi;
        this.teosed = teosed;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    public List getTeosed() {
        return teosed;
    }
    public void setTeosed(List teosed) {
        this.teosed = teosed;
    }
}
```

Koodinäide 21: Klass Autor

Uude konfiguratsioonifaili rida:

```
<mapping resource="Viies/Autor.hbm.xml"></mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class dynamic-insert="false" dynamic-update="false" mutable="true"
name="Viies.Autor"
  optimistic-lock="version" polymorphism="implicit" select-before-update="false"
table="AUTORID">
      <id name="id" type="int" column="ID">
        <generator class="native" />
      </id>
      <property name="nimi" column="NIMI" type="string"></property>
     <list name="teosed" table="AUTOR_TEOSED">
       <key column="AUTOR_ID"/>
       <list-index column="POSITION"/>
       <element type="string" column="PEALKIRI"/>
     </list>
  </class>
</hibernate-mapping>
```

Koodinäide 22: Autor.hbm.xml

```
package Viies;

import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Test1 {
    public static void main(String[] args) {
        SessionFactory sf = new
Configuration().configure("Viies/hibernate.cfg.xml").buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try {
            Autor a = new Autor();
            a.setId(1);
            a.setNimi("Karl Ristikivi");
            List l = new ArrayList();
            l.add("Hingede öö");
            l.add("Kahekordne mäng");
            a.setTeosed(l);
            s.save(a);
            tx.commit();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 23: Autor klassi testprogramm

Andmebaasis eksisteerib autor teostest eraldi, rakenduses on teosed klassi Autor osa:

```
mysql> select * from AUTORID;
+----+----------------+
| ID | NIMI           |
+----+----------------+
|  1 | Karl Ristikivi |
+----+----------------+
1 row in set (0.01 sec)

mysql> select * from AUTOR_TEOSED;
+----------+------------------+----------+
| AUTOR_ID | PEALKIRI         | POSITION |
+----------+------------------+----------+
|        1 | Hingede ├Č├Č      |        0 |
|        1 | Kahekordne m├żng  |        1 |
+----------+------------------+----------+
2 rows in set (0.02 sec)
```

Pilt 23: Tabelid AUTORID ja AUTOR_TEOSED

13.1 Ülesanne

1) Looge klass Tunniplaan, millel massiiv loengutega.

2) Looge testprogramm.

14 Relatsioonide tüübid

Kõik, kes kunagi tänapäeva andmebaasidega kokku puutunud, teavad, et seoseid eksisteerib põhimõtteliselt kolme tüüpi: üks-ühele, üks-mitmele ja mitu-mitmele. Seosed saab liigitada ka ühe- ja kahesuunalisteks. Mõistagi pakub Hibernate võimalust neid kirjeldada, kusjuures määrata saab ka, kas kasutatakse laiska laadimist ja kas kõiki seotuid kirjeid uuendatakse, kui üks muutub (*cascade*), mis tähendab, et rakenduse ressursikasutust on võimalik oluliselt optimiseerida.

14.1 Üks-ühele seos

Seos, mis tähendab, et üks on teise komponent. Näide on üsna sarnane eelnevalt vaadeldud Omaniku sidumisega Aadressiga. Kasutame annotatsioone.

```
package Kuues;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name="OPILASED")
public class Opilane implements java.io.Serializable {
    private int id;
    private String nimi;
    private Telefon telefon;
    public Opilane() {

    }
    public Opilane(String nimi, Telefon telefon) {
        this.nimi = nimi;
        this.telefon = telefon;
    }
    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="NIMI")
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    @OneToOne(cascade=CascadeType.ALL)
    public Telefon getTelefon() {
        return telefon;
    }
    public void setTelefon(Telefon telefon) {
        this.telefon = telefon;
    }
}
```

Koodinäide 24: Klass Opilane

Määrasime, et Telefon on seotud õpilasega ühesuunaliselt ja kui Telefon klassi objekti muudetakse, muudetakse ka seotud Opilane klassist objekti, kui vaja (CascadeType.ALL).

```
package Kuues;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="TELEFONINUMBRID")
public class Telefon {
    private int id;
    private String number;
    public Telefon() {

    }
    public Telefon(String number) {
        this.number = number;
    }
    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="NUMBER")
    public String getNumber() {
        return number;
    }
    public void setNumber(String number) {
        this.number = number;
    }
}
```

Koodinäide 25: Klass Telefon

Nagu ikka, tuleb klasside olemasolu ära märkida Hibernate konfiguratsioonifaili.

```
<mapping class="Kuues.Opilane"/>
<mapping class="Kuues.Telefon"/>
```

```
package Kuues;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

public class Test1 {
    public static void main(String[] kapsas) {
        AnnotationConfiguration c = new
AnnotationConfiguration().configure("Kuues/hibernate.cfg.xml");
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try
        {
            Telefon t = new Telefon("56999999");
            Opilane o = new Opilane("Oskar", t);
            s.save(o);
            tx.commit();
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 26: Üks-ühele testprogramm

Andmebaasist näeme, et Telefon klassist objekti kohta käiv kirje on seotud Opilane klassist objektiga võõrvõtme telefon_ID kaudu.

```
mysql> select * from telefoninumbrid;
+----+----------+
| ID | NUMBER   |
+----+----------+
|  1 | 56999999 |
+----+----------+
1 row in set (0.00 sec)

mysql> select * from opilased;
+----+-------+------------+
| ID | NIMI  | telefon_ID |
+----+-------+------------+
|  1 | Oskar |          1 |
+----+-------+------------+
1 row in set (0.00 sec)
```

Pilt 24: Üks-ühele seosega tabelid

14.2 Üks-mitmele seos

Eelmist näidet täiendades on lihtne luua realisatsiooni, kus igal õpilasel võib olla mitu telefoninumbrit. Klass Telefon jääb antud juhul samale kujule. Loome uue klassi Opilane2:

```
package Kuues;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="OPILASED2")
public class Opilane2 implements java.io.Serializable {
    private int id;
    private String nimi;
    private Set<Telefon> telefoninumbrid = new HashSet<Telefon>(0);
    public Opilane2() {

    }
    public Opilane2(String nimi, Set<Telefon> telefoninumbrid) {
        this.nimi = nimi;
        this.telefoninumbrid = telefoninumbrid;
    }
    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="NIMI")
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    @OneToMany(cascade=CascadeType.ALL)
    @JoinTable(name="OPILANE2_TELEFON",
joinColumns={@JoinColumn(name="OPILASED2_ID")},inverseJoinColumns={@JoinColumn(name="TELE
FONINUMBRID_ID")})
    public Set<Telefon> getTelefoninumbrid() {
        return telefoninumbrid;
    }
    public void setTelefoninumbrid(Set<Telefon> telefoninumbrid) {
        this.telefoninumbrid = telefoninumbrid;
    }
}
```

Koodinäide 27: Klass Opilane2

Õpilasega on seotud nüüd kogum telefoninumbreid. Selleks kasutasime *@OneToMany* annotatsiooni ja spetsifitseerisime, et tekitatakse tabel OPILANE2_TELEFON, mis seob telefoninumbreid õpilastega õpilase ID ja telefoni ID alusel.

Testprogrammis tuleb vaid ära muuta read, kus õpilasega seotakse telefoninumber:

```
Set<Telefon> telefoninumbrid = new HashSet<Telefon>();
telefoninumbrid.add(new Telefon("111111"));
telefoninumbrid.add(new Telefon("222222"));
Opilane2 o = new Opilane2("Oskar", telefoninumbrid);
```

Lisaks on tarvis importida *Set* ja *HashSet*.

Pärast testprogrammi käivitamist võime andmebaasis näha järgmiseid tabeleid:

```
mysql> select * from opilased2;
+----+-------+
| ID | NIMI  |
+----+-------+
|  1 | Oskar |
+----+-------+
1 row in set (0.00 sec)

mysql> select * from opilane2_telefon;
+--------------+--------------------+
| OPILASED2_ID | TELEFONINUMBRID_ID |
+--------------+--------------------+
|            1 |                  1 |
|            1 |                  2 |
+--------------+--------------------+
2 rows in set (0.00 sec)

mysql> select * from telefoninumbrid;
+----+--------+
| ID | NUMBER |
+----+--------+
|  1 | 111111 |
|  2 | 222222 |
+----+--------+
2 rows in set (0.00 sec)
```

Koodinäide 28: Üks-mitmele seos andmebaasis

Õpilane ID-ga 1 omab kahte telefoninumbrit, ID-dega 1 ja 2.

14.3 Mitu-mitmele seos

Selle seose realiseerimine on üsna sarnane eelmise näitega. Mõistagi ei eksisteeri relatsioonilises andmebaasist mitu mitmele seost kui sellist, vaid kasutatakse vahetabeleid.

```
package Kuues;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="Kuulajad")
public class Kuulaja implements java.io.Serializable {
    private int id;
    private String nimi;
    private Set<Kursus> kuulajakursused = new HashSet<Kursus>(0);
    public Kuulaja() {

    }
    public Kuulaja(String nimi, Set<Kursus> kuulajakursused) {
        this.nimi = nimi;
        this.kuulajakursused = kuulajakursused;
    }
    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="NIMI")
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(name="KUULAJA_KURSUS",
joinColumns={@JoinColumn(name="Kuulaja_ID")},inverseJoinColumns={@JoinColumn(nam
e="Kursus_ID")})
    public Set<Kursus> getKursused() {
        return kuulajakursused;
    }
    public void setKursused(Set<Kursus> kuulajakursused) {
        this.kuulajakursused = kuulajakursused;
    }
}
```

Koodinäide 29: Klass Kuulaja

```
package Kuues;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Kursus")
public class Kursus implements java.io.Serializable {
    private int id;
    private String nimi;
    private String semester;
    public Kursus() {

    }
    public Kursus(String nimi, String semester) {
        this.nimi = nimi;
        this.semester = semester;
    }
    @Id
    @GeneratedValue
    @Column(name="ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="NIMETUS")
    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    @Column(name="SEMESTER")
    public String getSemester() {
        return semester;
    }
    public void setSemester(String semester) {
        this.semester = semester;
    }
}
```

Koodinäide 30: Klass Kursus

```
package Kuues;

import java.util.HashSet;
import java.util.Set;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

public class Test3 {
    public static void main(String[] kapsas) {
        AnnotationConfiguration c = new
AnnotationConfiguration().configure("Kuues/hibernate.cfg.xml");
        SessionFactory sf = c.buildSessionFactory();
        Session s = sf.openSession();
        Transaction tx = s.beginTransaction();
        try
        {
            Set<Kursus> kursused = new HashSet<Kursus>();
            kursused.add(new Kursus("Rakenduste programmeerimine", "Sügis"));
            kursused.add(new Kursus("Graafika ja muusika programmeerimine",
"Sügis"));
            kursused.add(new Kursus("Portugali keel A1", "Sügis"));
            Kuulaja k1 = new Kuulaja("Lauri Elias", kursused);
            s.save(k1);
            tx.commit();
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage().toString());
            tx.rollback();
        }
        s.close();
    }
}
```

Koodinäide 31: Mitu mitmele testprogramm

```
mysql> select * from kuulajad;
+----+-------------+
| ID | NIMI        |
+----+-------------+
|  1 | Lauri Elias |
+----+-------------+
1 row in set (0.00 sec)

mysql> select * from kursus;
+----+---------------------------------------+----------+
| ID | NIMETUS                               | SEMESTER |
+----+---------------------------------------+----------+
|  1 | Portugali keel A1                     | Sügis    |
|  2 | Graafika ja muusika programmeerimine  | Sügis    |
|  3 | Rakenduste programmeerimine           | Sügis    |
+----+---------------------------------------+----------+
3 rows in set (0.00 sec)

mysql> select * from kuulaja_kursus;
+------------+-----------+
| Kuulaja_ID | Kursus_ID |
+------------+-----------+
|          1 |         1 |
|          1 |         2 |
|          1 |         3 |
+------------+-----------+
3 rows in set (0.00 sec)
```

Pilt 25: Mitu-mitmele tabelid

14.4 Ülesanne

1) Looge iga strateegia jaoks klassid.
2) Looge tervikrakendus, kus kõik seosed on kasutusel.

15 Hibernate ja Spring, integreerimine

Spring on Java raamistik, mida ei saa seostada konkreetse valdkonnaga. Tema eesmärk on hõlbustada arendust igal pool. Tuntumad Springi featuurid on AOP ja IoC. Raamistik pakub lisa abstraktsioonikihti rakenduse ja ORM komponentide, näiteks Hibernate, vahel. Springil puudub sisseehitatud püsiesituse raamistik. Spring võimaldab luua kergemini mõistetavat rakendust, mille loomiseks ei pea programmeerija kirjutama suurt hulka korduvat koodi.

15.1 DAO muster

Eelnevates näidetes kasutasime pidevalt Hibernate-i Session objekti, et teha püsiesitustoiminguid. DAO muster ütleb, et kõik sellised toimingud peaksid käima läbi kindla klassi, et kood oleks lihtsam ja hallatavam.

Lisame uuele projektile Spring Framework 2.5.6 ja Hibernate JPA. Et antud näidet kompileerida, läheb veel vaja kahte .jar faili järgnevatelt linkidelt:
http://commons.apache.org/dbcp/download_dbcp.cgi
http://commons.apache.org/pool/download_pool.cgi
Lisame ka need projekti *Libraries* kataloogi.

```
package HibernateSpring;

import java.util.Collection;

public interface SaadeDAO {
    public void lisaSaade(Saade saade);
    public void kustutaSaade(Saade saade);
    public Saade kysiSaade(int id);
    public Collection kysiSaated();
}
```

Koodinäide 32: Saade DAO

Järgmisena loome andmepääsu objekti liidese realisatsiooni.

```
package HibernateSpring;

import java.util.Collection;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

public class SaadeDAOImpl extends HibernateDaoSupport implements SaadeDAO {
    @Override
    public void lisaSaade(Saade saade) {
        getHibernateTemplate().save(saade);
    }
    @Override
    public void kustutaSaade(Saade saade) {
        getHibernateTemplate().delete(saade);
    }
    @Override
    public Saade kysiSaade(int id) {
        return (Saade)getHibernateTemplate().get(Saade.class, id);
    }
    @Override
    public Collection kysiSaated() {
        return getHibernateTemplate().find("from Saade");
    }
}
```

Koodinäide 33: Saade DAO realisatioon

```
package HibernateSpring;

public class Saade {
    private int id, kasolnud;
    private String pealkiri;
    public Saade() {

    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getKasolnud() {
        return kasolnud;
    }
    public void setKasolnud(int kasolnud) {
        this.kasolnud = kasolnud;
    }
    public String getPealkiri() {
        return pealkiri;
    }
    public void setPealkiri(String pealkiri) {
        this.pealkiri = pealkiri;
    }
}
```

Koodinäide 34: Klass Saade

Et Hibernate oskaks Saade klassist objekte jäädvustada, loome .hbm.xml faili:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="HibernateSpring.Saade" table="saated" catalog="mysql">
        <id name="id" type="java.lang.Integer">
            <column name="ID" />
            <generator class="identity" />
        </id>
        <property name="pealkiri" type="string">
            <column name="PEALKIRI" length="256" />
        </property>
        <property name="kasolnud" type="java.lang.Integer">
            <column name="KASOLNUD" />
        </property>
    </class>
</hibernate-mapping>
```

Koodinäide 35: Saade.hbm.xml

Kogu töö aga laseme ära teha Spring raamistikul. Seame üles vaid deklaratsioonifaili, Spring suhtleb ise Hibernatega ja loob vajalike objektide instantsid.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
">
    <bean id="andmebaas" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"><value>com.mysql.jdbc.Driver</value></property>
        <property name="url"><value>jdbc:mysql://localhost:3306/mysql</value></property>
        <property name="username"><value>root</value></property>
        <property name="password"><value>123</value></property>
    </bean>
    <bean id="sessioonivabrik"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref bean="andmebaas"/>
        </property>
        <property name="mappingResources">
            <list>
                <value>HibernateSpring/Saade.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">create</prop>
            </props>
        </property>
    </bean>
    <bean id="saadeDAO" class="HibernateSpring.SaadeDAOImpl">
        <property name="sessionFactory"><ref local="sessioonivabrik" /></property>
    </bean>
</beans>
```

Koodinäide 36: Springi ubade seadistusfail

Nagu ülalolevast failist näha, seadistame andmebaasi ilma Hibernate raamistiku .cfg.xml faili kasutamata. Järgmisena seome *SessionFactory* objekti andmebaasiga, teavitame seda Saade.hbm.xml kaardistusfaili olemasolust. Viimane uba seob sessioonivabriku DAO realisatsiooni klassiga.

```
package HibernateSpring;

import java.util.Collection;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test1 {
    public static void main(String[] kapsas) {
        ApplicationContext ac = new
ClassPathXmlApplicationContext("HibernateSpring/HibernateSpring.xml");
        SaadeDAO saadeDAO = (SaadeDAO)ac.getBean("saadeDAO");
        Saade s1 = new Saade();
        s1.setKasolnud(0);
        s1.setPealkiri("Suvereporter");
        Saade s2 = new Saade();
        s2.setKasolnud(0);
        s2.setPealkiri("Kaua võib");
        saadeDAO.lisaSaade(s1);
        saadeDAO.lisaSaade(s2);
        Collection nimekiri = saadeDAO.kysiSaated();
        for (Object o : nimekiri) {
            System.out.println(((Saade)o).getPealkiri());
        }
    }
}
```

Koodinäide 37: Spring + Hibernate test

15.2 Ülesanne

1) Looge nullist uued andmepääsuliidesed, klassid ja seadistused.
2) Looge testprogramm nende kontrollimiseks.

16 Prooviloeng

31. oktoobril 2011 viis autor läbi proovitunni, kus Rakenduste Programmeerimise aine üliõpilased katsetasid siinset õppematerjali. Tagasiside oli peamiselt neutraalne, suuremal osal üliõpilastest kadus huvi umbes pärast 2 tundi materjalidega töötamist, mis viitab sellele, et siinne töö on ühe korraga läbitöötamiseks liiga mahukas. Üliõpilased, kes juhendi otsast lõpuni läbisid, märkasid peamiselt vaid kirjavigu. Esines ka vigu ülesannete loogikas ja mõned tekstilõigud olid arusaamatud. Loodetavasti suutis autor suurema osa neist parandada. Kahel peatükil puudusid ka pildid andmetabelitest, autor lisas need.

17 Kokkuvõte

Õppematerjal tutvustas Hibernate raamistiku põhifunktsionaalsust. Tutvusime erinevate realisatsiooniviisidega – annotatsioonid ja deklaratsioonifailid. Põgusalt tutvustas autor NetBeans IDE ja andmebaasi paigaldamist lokaalsesse masinasse. Materjali lõpus tuli jutuks ka Spring raamistiku integreerimine Hibernate-ga. Töö alguses on kirjeldatud ORM-i olemust, relatsioonilise- ja objektorienteeritud maailma paradigmade kokkusobimatust, Hibernate-i alternatiive ning olemasolevaid võõrkeelseid tasuta õppematerjale internetis. Vaatlemisele tulid pärilus ja relatsioonide tüübid.

18 Kasutatud kirjandus

Karring, M.-L. (2005). Kasutamise kuupäev: 26. 10 2011. a., allikas java.andresteder.com: http://java.andresteder.com/static/java/works/Java_Veebiteenused_M-L_Karring.pdf

King, G., & Bauer, C. (2005). Java Persistence with Hibernate. Greenwich: Manning.

Raudjärv, R. (2005). Kasutamise kuupäev: 26. 10 2011. a., allikas scribd.com: http://www.scribd.com/doc/2316461/Inversion-of-Control-referaat

Seddighi, A. R. (2009). Spring Persistence with Hibernate. Birmingham: Packt Publishing Ltd.

Vilgota, A. (2005). Kasutamise kuupäev: 26. 10 2011. a., allikas quretec.com: http://www.quretec.com/u/vilo/edu/Students/Andres_Vilgota/semestritoo/semtoo.pdf