Object-Relational Mapping (ORM)

ORM, MyBatis, **Hibernate**, EclipseLink, JPA

Simon Károly
simon.karoly@codespring.ro

OOP ↔ RDB(MS)

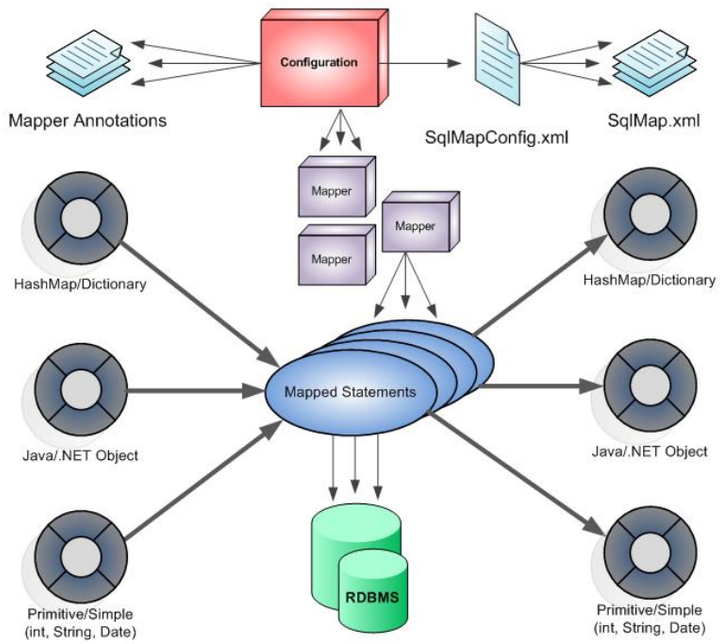
- Az OO és relációs „látásmódok” közötti eltérések (object-relational impedance mismatch):
  - Kettős séma (dual-schema problem)
  - Adatrejtés (encapsulation)
  - Öröklődés (inheritance)
  - Adattípusok, adatok manipulálása
  - Tranzakciók
  - Stb.
- Lehetőségek:
  - ORM
  - NoSQL / OODBMS
- Kérdések:
  - RDBMS vs. OODBMS
  - OODBMS vs. ORM
- Java ORM keretrendszerek: Hibernate, iBatis/MyBatis (ASF), OpenJPA (Apache), EclipseLink (Eclipse) (JPA reference implementation), TopLink (Oracle), stb.

iBatis → MyBatis

- Hat évig az Apache Software Foundation által támogatott, jelenleg a Google Code-on, illetve a GitHub-on elérhető Java, .NET és Ruby alapú fejlesztéseket támogató data mapper keretrendszer.
- A relációs adatbázisok (SQL) és objektumok közötti automatikus megfeleltetést (mappinget) valósítja meg.
- 2001 – Clinton Begin: iBatis, Secrets (kriptográfia)
- 2002 – JPetStore → iBatis DB Layer
- 2004 – ASF –nek adományozva
- 2010 – átviszik a Google Code-ra MyBatis néven (iBatis 3.0 → MyBatis 3.0)

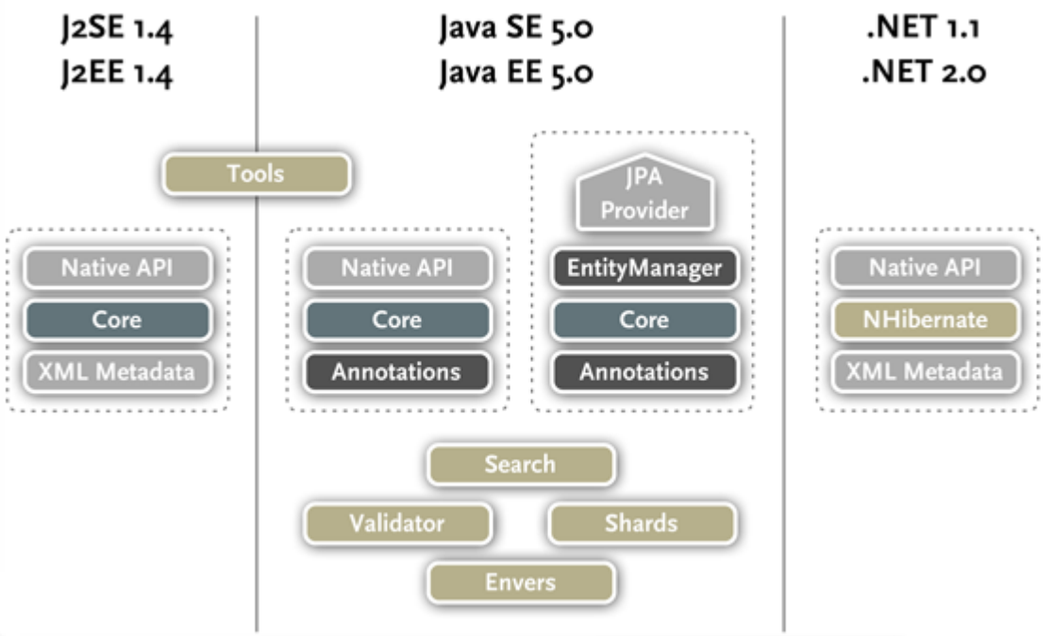
iBatis → MyBatis

- Java: az objektumok POJO-k.
- Mapper objektumokat alkalmaz.
- Az SQL statement-eket XML állományokba „csomagolja”, és jelentős mértékben csökkenti a kód méretét az alacsonyabb szintű API-khoz (pl. JDBC) képest, kényelmesebb (OO nézőpontból) adatmanipulációt téve lehetővé.
- Annotációk alkalmazása is lehetséges.
- Megközelítési mód (eltérések a különböző ORM rendszerek között):
  egy relációs adatbázisból kiindulva készül el az objektum modell (előnyösebb például, ha nincsen teljes hozzáférésünk az adatbázishoz).
- Ibator (MyBatis Generator): iBatis kódgenerátor (egy létező adatbázis alapján SqlMap XML állományokat, Java osztályokat (beaneket) és DAO-kat generál).
- http://ibatis.apache.org/ → www.mybatis.org



Hibernate

- Object-relational persistence and query service.
- JBoss (Red Hat), nyílt forráskód (LGPL).
- Hibernate Core (including annotations), Hibernate EntityManager (JPA), további opcionális modulok.
- Osztályok: egyedi azonosítóval és alapértelmezett konstruktorral rendelkező POJOk (reflection alkalmazása).
- Queries: HSQL, SQL, OO Criteria and Example API.



Hibernate – „getting started”

- 1. lépés:
  - Hibernate disztribúció letöltése (www.hibernate.org).
  - Opcionális: Hibernate Tools Eclipse plug-in telepítése (a JBoss Tools része → telepítés az ennek megfelelő update site-ról).
- 2. lépés: megfelelő csomagok (.jar állományok) hozzárendelése a projekthez:
  - Hibernate Core csomag.
  - Felhasznált csomagok (min. a Hibernate disztribúcióban „required”-ként feltüntetett .jar állományok).
  - Adatbázis kapcsolatért felelős csomag(ok) (pl. MySql Connector).
  - Naplózásért felelős csomag (pl. log4j) + megfelelő slf4j bridge (pl. slf4j-log4j).
  - Javaslat: csomagok frissítése.
- 3. lépés: adatbázis táblák és a perzisztens objektumoknak megfelelő osztályok (automatikus sémagenerálásra is lehetőség van).
- 4. lépés: hibernate konfigurációs állomány és mapping állományok létrehozása (xml) (annotációk is alkalmazhatóak).
- 5. lépés: SessionFactory létrehozása.
- 6. lépés: DAO-k implementációja

Hibernate – Event.java

```
import java.util.Date;

public class Event {

    private Long id;
    private String title;
    private Date date;

    public Event () {}

    public Long getId () {
       return id;
    }
    private void setId (Long id) {
       this.id = id;
    }
    public Date getDate () {
       return date;
    }
    public void setDate (Date date) {
       this.date = date;
    }
    public String getTitle () {
       return title;
    }
    public void setTitle (String title) {
       this.title = title;
    }
}
```

Megjegyzések:

- POJO
- JavaBean névkonvenciók alkalmazása (javasolt, nem kötelező)
- A Hibernate közvetlenül is hozzá tud férni a mezőkhöz
- Argumentumok nélküli konstruktor (példányosítás: reflection) (private is lehet)
- Egyedi azonosító (id mező) (**private setter** – később nem módosítható) (a Hibernate hozzáfér a private/public/ protected mezőkhöz és metódusokhoz)

Hibernate configuration

- hibernate.cfg.xml állomány létrehozása a projektnek megfelelő classpath gyökerében.
- A konfigurációs állomány tartalma: a SessionFactory beállításai: egy session factory egy adott adatbázis kapcsolatért felelős. Amennyiben többet használunk több ilyen tag-et kell létrehoznunk (javaslat: különböző konfigurációs állományok):
  - Adatbázis kapcsolat beállítása (driver, url, user, password).
  - Mapping állományok megadása.
  - Egyéb beállítások:
    - SQL dialect beállítása (milyen típusú query-ket generáljon a Hibernate) (HSQL, MySQL5Dialect stb.).
    - Session managemenet beállítása (pl. thread/jta).
    - Schema managemenet (hbm2dll.auto, validate/update/create/create-drop).
    - Caching mechanizmus.
    - Kiírt üzenetek szűrése (pl. sql queryk kiírása konzolra).
    - stb.

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
      <!-- Database connection settings -->
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost:3306/test</property>
      <property name="connection.username">root</property>
      <property name="connection.password"></property>
      <!-- JDBC connection pool (use the built-in) -->
      <property name="connection.pool_size">1</property>
      <!-- SQL dialect -->
      <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
      <!-- Enable Hibernate's automatic session context management -->
      <property name="current_session_context_class">thread</property>
      <!-- Disable the second-level cache -->
      <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">true</property>
      <!-- Drop and re-create the database schema on startup -->
      <property name="hbm2ddl.auto">create</property>
      <mapping resource="event_map.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Event_map.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="core.Event" table="EVENTS">
      <id name="id" column="EVENT_ID">
                <generator class="native"/>
      </id>
      <property name="date" type="timestamp" column="EVENT_DATE"/>
      <property name="title"/>
    </class>

</hibernate-mapping>
```

Hibernate - SessionFactory

- SessionFactory létrehozása:
  HibernateUtil.java:

```
import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
      try {
        // Create the SessionFactory from hibernate.cfg.xml
        sessionFactory = new Configuration ().configure ().buildSessionFactory ();
      } catch (Throwable ex) {
        // Make sure you log the exception, as it might be swallowed
        System.err.println ("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError (ex);
      }
    }

    public static SessionFactory getSessionFactory () {
      return sessionFactory;
    }
}
```

Adatok feldolgozása

- Create/insert:

```
private void createAndStoreEvent (String title, Date theDate) {
    Session session = HibernateUtil.getSessionFactory ().getCurrentSession ();
    session.beginTransaction ();
    Event theEvent = new Event ();
    theEvent.setTitle (title);
    theEvent.setDate (theDate);
    session.save (theEvent);
    session.getTransaction ().commit ();
}
```

- List:

```
private List listEvents () {
    Session session = HibernateUtil.getSessionFactory ().getCurrentSession ();
    session.beginTransaction ();
    List result = session.createQuery ("from Event").list ();
    session.getTransaction ().commit ();
    return result;
}
```

Adatok feldolgozása

- Inserthez (save-hez) hasonlóan alkalmazható:

```
session.delete (object);
session.update (object);
Event e = (Event) session.load (Event.class, eventId);
```

Vagy:

```
Event e = (Event) session.get (Event.class, eventId);
```

(a load kivételt dobhat, a get null értéket téríthet vissza)

- A metódusok meghívása:

```
EventManager mgr = new EventManager ();
mgr.createAndStoreEvent ("My Event", new Date ());
List events = mgr.listEvents ();
```

- Végül:

```
HibernateUtil.getSessionFactory ().close ();
```

Hibernate - Associations

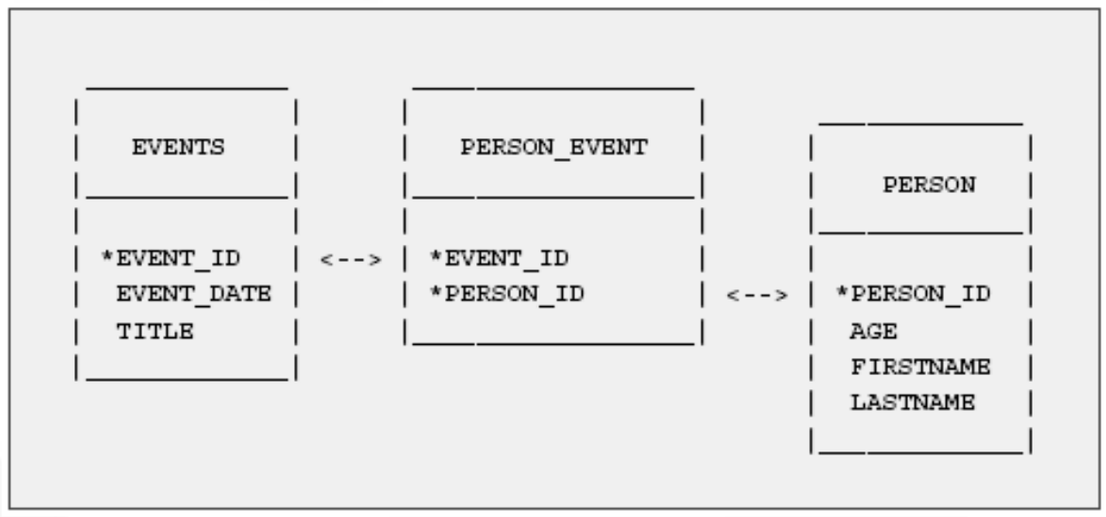
- Person.java:

```
public class Person {

    private Long id;
    private int age;
    private String firstname;
    private String lastname;
    private Set events = new HashSet();

    public Person() {}
    …// getters and setters

}
```



Hibernate - Associations

- Mapping (set-based many to many association):

```
<class name="events.Person" table="PERSON">
    <id name="id" column="PERSON_ID">
                <generator class="native"/>
    </id>
    <property name="age"/>
    <property name="firstname"/>
    <property name="lastname"/>
    <set name="events" table="PERSON_EVENT">
      <key column="PERSON_ID"/>
      <many-to-many column="EVENT_ID" class="events.Event"/>
    </set>
</class>
```

- Használat:

```
session.beginTransaction ();
Person aPerson = (Person) session.load (Person.class, personId);
Event anEvent = (Event) session.load (Event.class, eventId);
aPerson.getEvents ().add (anEvent);
session.getTransaction ().commit ();
```

Criteria queries

- A session egyben egy Criteria Factory is (org.hibernate.criterion.Criterion interfész implementációk):

```
Criteria crit = sess.createCriteria (Cat.class);
crit.setMaxResults (50);
List cats = crit.list ();
```

- A org.hibernate.criterion.Restrictions osztály Factory beépített Criteria típusok részére:

```
List cats = sess.createCriteria (Cat.class)
      .add( Restrictions.like ("name", "Fritz%") )
      .add( Restrictions.or (
                Restrictions.eq ( "age", new Integer (0) ),
                Restrictions.isNull ("age")
      ) ).list();
```

- SQL alkalmazása:

```
List cats = sess.createCriteria (Cat.class)
    .add( Restrictions.sqlRestriction ("lower({alias}.name) like
    lower(?)", "Fritz%", Hibernate.STRING) ).list( );
```

Criteria queries

- Ordering:

```
List cats = sess.createCriteria (Cat.class)
    .add (Restrictions.like ("name", "F%")
    .addOrder (Order.asc ("name"))
    .addOrder (Order.desc ("age"))
    .setMaxResults (50)
    .list ();
```

- Associations:

```
List cats = sess.createCriteria (Cat.class)
    .add(Restrictions.like ("name", "F%"))
    .createCriteria ("kittens")
    .add (Restrictions.like("name", "F%")).list ();
```

- Example alkalmazása:

```
Cat cat = new Cat ();
cat.setSex ('F');
cat.setColor (Color.BLACK);
List results = session.createCriteria(Cat.class)
    .add(Example.create(cat)).list ();
```

Criteria queries

- Projections:

```
List results = session.createCriteria (Cat.class)
    .setProjection (Projections.projectionList ()
      .add (Projections.rowCount ())
      .add (Projections.avg ("weight"))
      .add (Projections.max ("weight"))
      .add (Projections.groupProperty ("color")))
    .list ();
```

- Detached queries:

```
DetachedCriteria query = DetachedCriteria.forClass (Cat.class)
    .add (Property.forName ("sex").eq ('F'));
Session session = ....;
Transaction txn = session.beginTransaction ();
List results =
    query.getExecutableCriteria (session).setMaxResults (100).list ();
txn.commit ();
session.close ();
```

- És továbbá lehetséges:

```
sess.createSQLQuery ("SELECT * FROM CATS").list();
```

JPA

- Standard (JSR-317): perzisztencia (általában objektumok adatbázisba történő leképezése) – egységesített eljárás meghatározása.
- Specifikáció: Java Persistence API (JPA) (az EJB 3.0-tól különálló).
- Objektumok állapotának relációs adatbázisban történő tárolásának szempontjából a JDBC feletti absztrakciós szint.
- Entitiy Beans: POJOk, amelyek a JPA meta-adatok (annotációs mechanizmus) segítségével le lesznek képezve egy adatbázisba. Az adatok mentése, betöltése, módosítása megtörténhet anélkül, hogy a fejlesztőnek ezzel kapcsolatos kódot kelljen írnia (pl. JDBC hozzáféréssel kapcsolatos kód nem szükséges).
- A JPA meghatároz egy lekérdező nyelvet is (a funkcionalitások azonosak az SQL nyelvek által biztosított funkcionalitásokkal, de Java objektumokkal dolgozhatunk, az objektumorientált szemléletmódnak megfelelően).
- A JPA specifikáció meghatároz egy teljes ORM leképezést, a komponensek hordozhatóak (a mechanizmus már nem függ gyártótól, vagy alkalmazásszerver típustól) és hagyományos Java (SE) alkalmazásokon belül is használhatóak.
- Implementációk: EclipseLink (referencia), Hibernate, TopLink, OpenJPA stb.

Entity Beans

- POJOk, üzleti logikával kapcsolatos, "főnevek" által meghatározható entitások reprezentációi (kliens, raktári tárgy, hely, felszerelés stb.) (modell osztályok).
- Elsődleges kulcs (primary key): azonosítja a bean objektumot a memóriában, és ugyanakkor egy egyedi azonosítóként szolgál a megfelelő adatbázis bejegyzés számára.
- Bean osztály (bean class): perzisztens adatok reprezentációja objektumok segítségével. Ezen kívül, bizonyos esetekben tartalmazhat vonatkozó, üzleti logikával kapcsolatos kódot (validáció stb.) de ez nem jellemző. POJO, nem kell semmilyen interfészt megvalósítania és szerializálhatónak sem kell lennie.
  - A @javax.presistence.Entity annotációval kell megjelölni, kell tartalmaznia egy mezőt, vagy getter metódust az elsődleges kulcs részére, amelyet a @javax.persistence.Id annotáció jelöl. Ezen kívül rendelkezésünkre állnak további annotációk, amelyekkel a teljes ORM leképezés megvalósítható.

Entity Beans

```
package dev.com.titan.domain;

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

@Entity
@Table (name = "CABIN")
public class Cabin implements java.io.Serializable {

        private static final long serialVersionUID = 1L;
        private int id;
        private String name;
        private int deckLevel;
        private int shipId;
        private int bedCount;

        @Id
        @Column (name = "ID")
        public int getId () { return id; }
        public void setId (int id) { this.id = id;}
```

Entity Beans

```
@Column(name="NAME")
public String getName() { return name; }
public void setName(String name) { this.name = name; }

@Column(name="DECK_LEVEL")
public int getDeckLevel() { return deckLevel; }
public void setDeckLevel(int deckLevel) { this.deckLevel = deckLevel; }
```

```
    @Column(name="SHIP_ID")
    public int getShipId() { return shipId; }
    public void setShipId(int shipId) { this.shipId = shipId; }

    @Column(name="BED_COUNT")
    public int getBedCount() { return bedCount; }
    public void setBedCount(int bedCount) { this.bedCount = bedCount; }

    public static long getSerialversionuid() { return serialVersionUID; }

    public String toString() { return id + " " + name; }

}
```

Entity Manager

```
@PersistenceContext(unitName="titan") EntityManager entityManager;

@TransactionAttribute(REQUIRED)
public Cabin someMethod() {
        Cabin cabin = entityManager.find(Cabin.class, 1);
        cabin.setName("new name");
        return cabin;
}
```

EclipseLink

- Open-source Java perzisztencia keretrendszer
- Alapja az Oracle TopLink keretrendszere, az Eclipse Foundation projektje
- Teljes mértékben implementálja a JPA (referencia implementáció) és JAXB (Java Architecture for XML Binding) specifikációkat, ezen kívül tartalmazza az EclipseLink Database Web Services és EclipseLink Enterprise Information Services (JCA-n keresztüli adathozzáférés) részeket.
- Támogatja bizonyos NoSQL adatbázisokhoz a JPA hozzáférést (MongoDB stb.)
- Több alkalmazásszerverbe integrált/integrálható: Glassfish, JBoss stb. + több web szerver (Tomcat stb.) és a Gemini JPA projekt által OSGi környezetben is használható (OSGi JPA)

BiblioSpring – Hibernate DAL

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <!-- Database connection settings will be added programatically.
             They will be provided by the PropertyProvider. -->

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>
        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Schema management (validate/update/create/create-drop -->
        <property name="hbm2ddl.auto">update</property>

        <mapping
            resource="edu/codespring/bibliospring/backend/repository/hibernate/mapping/Title.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

BiblioSpring – Hibernate DAL

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="edu.codespring.bibliospring.backend.model">

<class name="Title" table="Title">
    <id name="id" type="long" column="id">
        <generator class="native"/>
    </id>
    <discriminator column="TYPE" type="string"/>
    <property name="title" column="title"/>
    <property name="isbn"/>
    <property name="publishingDate"/>

    <many-to-one name="editor" class="Editor" column="Editor_id" lazy="false"></many-to-one>

    <!-- Book extends Title -->
    <subclass name="Book" discriminator-value="BOOK">
        <!-- There is a many-to-many relationship between books and authors -->

        <bag name="authors" table="TitleAuthor">
        <key column="Title_id"/>
        <many-to-many column="Author_id" class="Author" lazy="false"/>
    </bag>

    </subclass>
</class>
```

BiblioSpring – Hibernate DAL

```
<class name="Author">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="firstName"/>
        <property name="lastName"/>
        <bag name="books" inverse="true" table="TitleAuthor">
        <key column="Author_id"/>
        <many-to-many column="Title_id" class="Book"/>
    </bag>
</class>

<class name="Editor">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="name"/>
        <property name="info"/>
        <bag name="titles" table="Title" inverse="true" lazy="true" fetch="select">
            <key>
                <column name="Editor_id" not-null="true" />
            </key>
            <one-to-many class="Title" />
        </bag>

</class>


</hibernate-mapping>
```

BiblioSpring – Hibernate DAL

```
package edu.codespring.bibliospring.backend.repository.hibernate;

//…imports

public class SessionManager {

    private static final Logger         LOG            = LoggerFactory.getLogger (SessionManager.class);
    private static final SessionFactory sessionFactory = buildSessionFactory ();

    private static SessionFactory buildSessionFactory () {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            // Setup connection using the PropertyProvider
            final Configuration configuration = new Configuration ();
            configuration.configure ();
            configuration.setProperty ("hibernate.connection.username", PropertyProvider.INSTANCE.getProperty ("dbUser"));
            configuration.setProperty ("hibernate.connection.password", PropertyProvider.INSTANCE.getProperty ("dbPassword"));
            configuration.setProperty ("hibernate.connection.url", PropertyProvider.INSTANCE.getProperty ("dbURL"));
            configuration.setProperty ("hibernate.connection.pool_size",
                    PropertyProvider.INSTANCE.getProperty ("dbConnectionPoolSize"));
            final ServiceRegistry serviceRegistry = new ServiceRegistryBuilder ().applySettings (
                    configuration.getProperties ()).buildServiceRegistry ();
            return configuration.buildSessionFactory (serviceRegistry);
        } catch (final Throwable ex) {
            LOG.error ("SessionFactory initialization failed", ex);
            throw new ExceptionInInitializerError (ex);
        }
    }

    public static SessionFactory getSessionFactory () {
        return sessionFactory;
    }

}
```

BiblioSpring – Hibernate DAL

```
package edu.codespring.bibliospring.backend.repository.hibernate;

import java.util.List;

import org.hibernate.Criteria;

public class HibernateDAO<T> {

    @SuppressWarnings ("unchecked")
    public List<T> getQueryResult (final Criteria q) {
        final List<T> list = q.list ();
        return list;
    }

}
```

BiblioSpring – Hibernate DAL

```
package edu.codespring.bibliospring.backend.repository.hibernate;

//…imports

public class HibernateBookDAO extends HibernateDAO<Book> implements BookDAO {

…

    @Override
    public List<Book> getBooksByFilter (final String pattern) throws RepositoryException {
        List<Book> bookList = Collections.emptyList ();
        Session session = null;
        try {
            session = SessionManager.getSessionFactory ().getCurrentSession ();
            session.beginTransaction ();
            final Criteria c = session.createCriteria (Book.class).add (
                    Restrictions.ilike ("title", "%" + pattern + "%"));
            bookList = this.getQueryResult (c);
            for (final Book b : bookList) {
                Hibernate.initialize (b.getAuthors ());
            }
            session.getTransaction ().commit ();
        } catch (final HibernateException ex) {
            if (session != null && session.getTransaction () != null) {
                session.getTransaction ().rollback ();
            }
            LOG.error ("Book selection failed", ex);
            throw new RepositoryException ("Book selection failed");
        }
        return bookList;
    }
```

BiblioSpring – Hibernate DAL

```
...

    @Override
    public void insertBook (final Book book) throws RepositoryException {
        Session session = null;
        try {
            session = SessionManager.getSessionFactory ().getCurrentSession ();
            session.beginTransaction ();
            session.persist (book);
            session.getTransaction ().commit ();
        } catch (final HibernateException ex) {
            if (session != null && session.getTransaction () != null) {
                session.getTransaction ().rollback ();
            }
            LOG.error ("Book insertion failed", ex);
            throw new RepositoryException ("Book insertion failed");
        }
    }

...

}
```