Object-Relational Mapping (ORM)
and
Hibernate
Problem area

- When working with object-oriented systems, there’s a mismatch between the object model and the relational database.

- How do we map one to the other?

```java
public class Student {
    private String name;
    private String address;
    private Set<Course> courses;
    private Set<Degree> degrees;
}
```

Java object with properties and associations

Relational database with tables and columns
Problem area

• How to map associations between objects?
  – References are directional, foreign keys not
  – Foreign keys can’t represent many-to-many associations
Technology

• Why relational databases?
  – Flexible and robust approach to data management
  – De-facto standard in software development

• Why object-oriented models?
  – Business logic can be implemented in Java (opposed to stored procedures)
  – Allows for use of design patterns and concepts like polymorphism
  – Improves code reuse and maintainability

• Demand for mapping interaction!
Approaches to ORM

• Write SQL conversion methods by hand using JDBC
  – Tedious and requires lots of code
  – Extremely error-prone
  – Non-standard SQL ties the application to specific databases
  – Vulnerable to changes in the object model
  – Difficult to represent associations between objects

```java
public void addStudent( Student student )
{
    String sql = "INSERT INTO student ( name, address ) VALUES ( " +
        student.getName() + ", " + student.getAddress() + " );"

    // Initiate a Connection, create a Statement, and execute the query
}
```
Approaches to ORM

• Use Java serialization – write application state to a file
  – Can only be accessed as a whole
  – Not possible to access single objects

• Object oriented database systems
  – No complete query language implementation exists
  – Lacks necessary features
The preferred solution

- Use an Object-Relational Mapping System (e.g., Hibernate)
- Provides a simple API for storing and retrieving Java objects directly to and from the database
- Non-intrusive: No need to follow specific rules or design patterns
- Transparent: Your object model is unaware

![Diagram of ORM/Hibernate with Student, Course, Degree, ORM/Hibernate, and Relational database connected by arrows labeled "Magic happens here!".]
Side step - preferred solution?

• Well, at topic of debate… NoSQL
• Pushed mostly by “really big” use cases
• Hits limits in traditional RDMS’s
• Hardware has changed (multicore, lots of cheap RAM)
• Don’t necessarily need RDMS’s traditional properties
  ▪ E.g. realtime more important than (guaranteed) consistency
NoSQL

- Not one type of solution
  - Document oriented
  - Graph dbs
  - Key-value stores
- NoSQL - Not Only SQL
- Interesting – but not “mainstream”
  - at least not yet
ORM and Architecture

- Middleware that manages persistence
- Provides an abstraction layer between the domain model and the database
Example app: The EventManager

Diagram:
- Java objects
- Hibernate mapping files
- Hibernate API
- Hibernate configuration
Java objects (POJO)

- Identifier property
- No-argument constructor
- Follows the JavaBean naming conventions

```java
public class Event {
    private int id;
    private String title;
    private Date date;
    private Set<Person> persons = new HashSet<Person>();

    public Event() {
    }

    public int getId() {
        return id;
    }

    private void setId( int id ) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle( String title ) {
        this.title = title;
    }

    // Getter and setter for date and persons
}
```
Example app: The EventManager
Hibernate mapping files

- Tells Hibernate which tables and columns to use to load and store objects

```
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="no.uio.inf5750.Event" table="events">
    <id name="id" column="event_id">
      <generator class="native"/>
    </id>
    <property name="title" not-null="true" unique="true"/>
    <property name="date" type="date" column="event_date"/>
    <set name="persons" table="event_persons">
      <key column="event_id"/>
      <many-to-many column="person_id" class="no.uio.inf5750.example.model.Person"/>
    </set>
  </class>
</hibernate-mapping>
```

Filename: Event.hbm.xml
Property mapping

The name property refers to the get/set-methods

Title must be not null and unique

...<property name="title" not-null="true" unique="true"/>
<property name="date" type="Date" column="event_date"/>
...

Types are Hibernate mapping types. Hibernate will guess if no type is specified.

Property name used as default if no column is specified
The name property refers to the get/set-methods.

Many-to-many associations require a link table.

Column name for "this" side of association

Column name for "other" side of association

Reference to the associated class

...  <set name="persons" table="event_persons">
    <key column="event_id"/>
    <many-to-many column="person_id"
      class="no.uio.inf5750.example.model.Person"/>
  </set>
...
Hibernate mapping types

- Hibernate will translate Java types to SQL / database types for the properties of your mapped classes

<table>
<thead>
<tr>
<th>Java type</th>
<th>Hibernate type</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>string</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>java.util.Date</td>
<td>date, time</td>
<td>DATE, TIME</td>
</tr>
<tr>
<td>java.lang.Integer, int</td>
<td>integer</td>
<td>INT</td>
</tr>
<tr>
<td>java.lang.Class</td>
<td>class</td>
<td>varchar</td>
</tr>
<tr>
<td>java.io.Serializable</td>
<td>serializable</td>
<td>BLOB, BINARY</td>
</tr>
</tbody>
</table>
Example app: The EventManager

- Java objects
- Hibernate mapping files
- Hibernate configuration
- Hibernate API
Hibernate configuration

• Each database has a *dialect*
  – hibernate.dialect = org.hibernate.dialect.H2Dialect

• Must also specify:
  – JDBC driver class
  – Connection URL
  – Username
  – Password

• More later in the lecture...
Example app: The EventManager

- Java objects
- Hibernate mapping files
- Hibernate API
- Hibernate configuration
The SessionFactory interface

- Provides Session instances to the application
- Shared among application threads
- Most important method is `getCurrentSession`
The Session interface

• Obtained from a SessionFactory
• Main runtime interface between a Java application and Hibernate
• Responsible for storing and retrieving objects
• Think of it as a collection of loaded objects related to a single unit of work

```
Session session = sessionFactory.getCurrentSession();
```
Instance states

• An object instance state is related to the persistence context
• The persistence context = a *Hibernate Session* instance
• Three types of instance states:
  – Transient
    • The instance is *not* associated with any persistence context
  – Persistent
    • The instance is associated with a persistence context
  – Detached
    • The instance was associated with a persistence context which has been closed – currently *not* associated
The Session interface

- Make a transient object persistent

  ```java
  Event event = new Event( "title", new Date() );
  Integer id = (Integer) session.save( event );
  ```

- Load an object – if matching row exists

  ```java
  Event event = (Event) session.load( Event.class, id );
  ```

- Load an object – if unsure about matching row

  ```java
  Event event = (Event) session.get( Event.class, id );
  ```

- Delete an object – make it transient again

  ```java
  session.delete( event );
  ```
The Session interface

- Update an object – if its detached
  
  `session.update(event);`

- Update or save an object – if you’re unsure about the state
  
  `session.saveOrUpdate(event);`

- Synchronize database with persistence context
  
  `session.flush(); // Happens auto. at transaction.commit()`
The Criteria interface

• You need a *query* when you don’t know the identifiers of the objects you are looking for
• Criteria used for *programmatic* query creation

Retrieve all instances of Event

Criteria criteria = session.createCriteria( Event.class );
List events = criteria.list();

Narrow the result set

Criteria criteria = session.createCriteria( Event.class );
criteria.add( Restrictions.eq( "title", "Rolling Stones" ) );
criteria.add( Restrictions.gt( "date", new Date() ) );
criteria.setMaxResults( 10 );
List events = criteria.list();
Transactions

- Transaction: A set of database operations which must be executed in entirety or not at all
- Should end either with a *commit* or a *rollback*
- All communication with a database has to occur inside a transaction!
Hibernate in real-life apps

• Spring used for SessionFactory management
  – Spring has excellent ORM integration
  – Custom SessionFactory management is “boilerplate-code”

• Spring used for Transaction management
  – Custom tx management is error prone
  – Support for declarative tx management with annotations
  – Consistent programming model across JTA, JDBC, Hibernate

• Annotate transactional methods / class with @Transactional
Spring-Hibernate dependencies

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>3.5.1-Final</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>3.5.1-Final</version>
</dependency>

<dependency>
    <groupId>geronimo-spec</groupId>
    <artifactId>geronimo-spec-jta</artifactId>
    <version>1.0-M1</version>
</dependency>

<dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
</dependency>

<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.5.8</version>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.2.136</version>
</dependency>
Spring-Hibernate configuration

```xml
<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
    <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list>
            <value>hibernate/Event.hbm.xml</value>
            <value>hibernate/Person.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.H2Dialect</prop>
            <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
        </props>
    </property>
</bean>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
    <property name="driverClass" value="org.h2.Driver"/>
    <property name="jdbcUrl" value="jdbc:h2:mem:eventmanager;DB_CLOSE_ON_EXIT=FALSE"/>
    <property name="user" value="sa"/>
    <property name="password" value=""/>
</bean>
```

- Enables annotations
- Can be injected like any bean
- Points to mapping files
- Hibernate properties
- Connection properties
Example: The EventManager

- Java objects
- Hibernate mapping files
- Hibernate API
- Hibernate configuration
Advantages of ORM

• **Productivity**
  – Eliminates lots of repetitive code – focus on business logic
  – Database schema is generated automatically

• **Maintainability**
  – Fewer lines of code – easier to understand
  – Easier to manage change in the object model
Advantages of ORM

• Performance
  – Lazy loading – associations are fetched when needed
  – Caching

• Database vendor independence
  – The underlying database is abstracted away
  – Can be configured outside the application
Resources

• Spring documentation chapter 13

• Hibernate reference documentation
  – www.hibernate.org -> Documentation -> Reference