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

```java
public class Student {
    private Collection<Degree> degrees;
    ...
}
```
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 a *Object-Relational Mapping System* (eg. 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](image)
ORM and Architecture

- Middleware that manages persistence
- Provides an abstraction layer between the domain model and the database

Diagram:
- Presentation Layer
  - Service/Business Layer
  - Persistence Layer
  - ORM / Hibernate
  - (Database)
  - Domain Model
Example app: The EventManager
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

```xml
<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.

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

Reference to the associated class.

Column name for "this" side of association.

Column name for "other" side of association.
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

Diagram:

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

<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="hibernate.connection.url">jdbc:hsqldb:hsql://localhost</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <property name="hibernate.connection.pool_size">10</property>
        <property name="hibernate.hbm2ddl.auto">create-drop</property>
        <mapping resource="Event.hbm.xml"/>
        <mapping resource="Person.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

Filename: hibernate.cfg.xml
Example app: The EventManager

Java objects

Hibernate mapping files

Hibernate configuration file

Hibernate API
The Configuration class

- Represents a set of mapping files
- Mapping files can be specified programmatically or through the Hibernate configuration file
- Intended as a startup-time object

```java
Configuration configuration = new Configuration()
.addResource( "Event.hbm.xml" )
.addResource( "Person.hbm.xml" );
```

...or...

```java
Configuration configuration = new Configuration();
configuration.configure();
```

Loads Hibernate.cfg.xml from the classpath
The SessionFactory interface

- Obtained from a Configuration instance
- Shared among application threads
- Main purpose is to provide Session instances
- Allowed to instantiate more than one SessionFactory
- Sophisticated implementation of the factory design pattern

SessionFactory sessionFactory = configuration.buildSessionFactory();
The Session interface

- Obtained from a SessionFactory instance
- 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

```java
Session session = sessionFactory.openSession();
```
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

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

Load an object – if matching row exists

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

Load an object – if unsure about matching row

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

Delete an object – make it transient again

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!
Transactions

- Most common pattern is session-per-request

```java
Session session = sessionFactory.openSession();
Transaction transaction = null;

try {
    transaction = session.beginTransaction();
    session.save(event);
    session.save(person);
    transaction.commit();
} catch (RuntimeException ex) {
    if (transaction != null) {
        transaction.rollback();
    }
    throw ex;
} finally {
    session.close();
}
```
Example: The EventManager

```
Java objects
```

```
Hibernate mapping files
```

```
Hibernate API
```

```
Hibernate configuration file
```

Diagram showing the relationship between Java objects, Hibernate mapping files, Hibernate API, and Hibernate configuration file.
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

• Books on Hibernate
  – Christian Bauer and Gavin King: *Hibernate in Action*
  – James Elliot: *Hibernate – A Developer’s notebook*
  – Justin Gehtland, Bruce A. Tate: *Better, Faster, Lighter Java*

• The Hibernate reference documentation
  – [www.hibernate.org](http://www.hibernate.org)