



Software Structures and Models

Hibernate

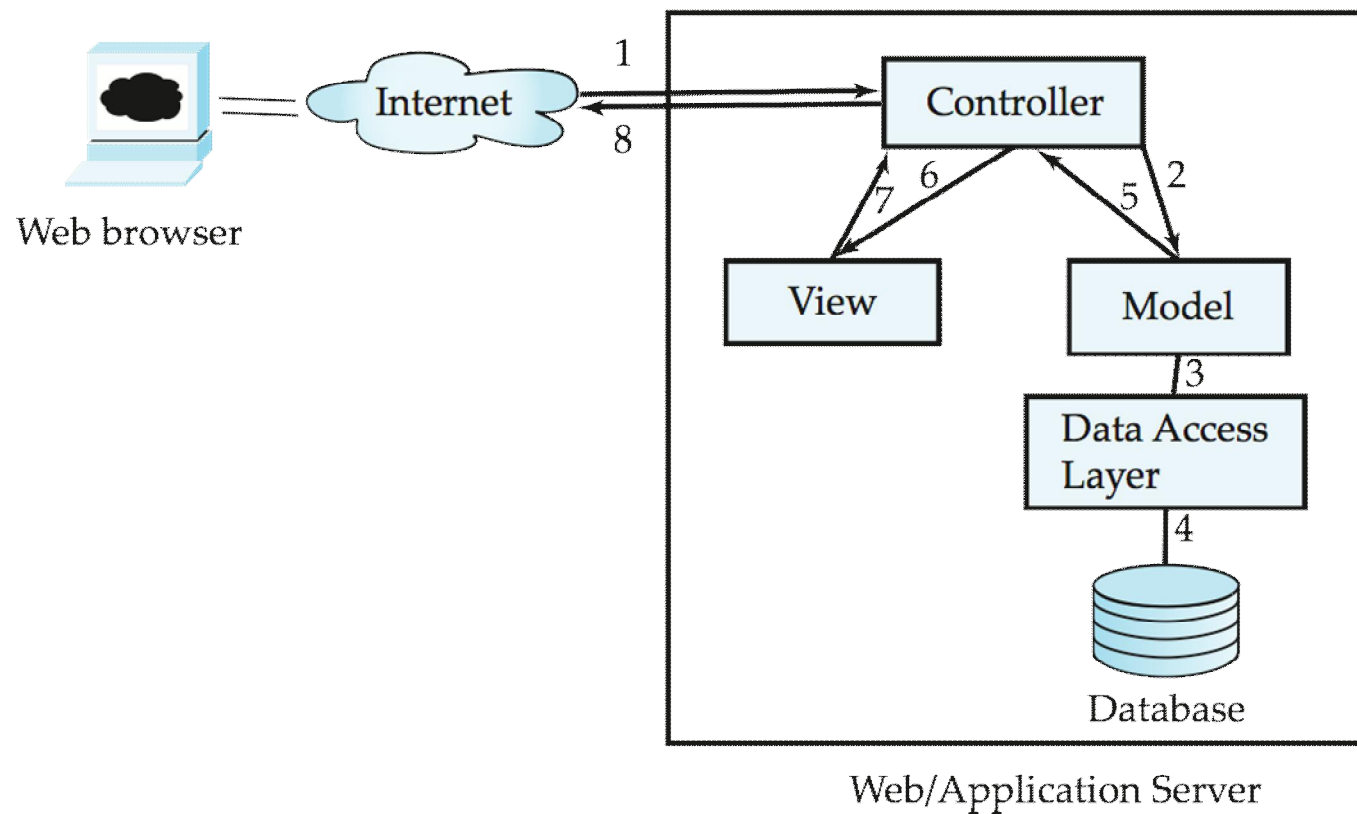


Object relational mapping

Acknowledgements

In the following Vesa Ollikainen's course material has been used as a source of information as well as the material in Hibernate tutorial in <http://www.tutorialspoint.com/hibernate/index.htm>

A typical application architecture



Lähde: Silberschatz & al.



Persistence

- Why are (relational) databases needed in object oriented programming?
- Why do we use 45 years old ideas together with modern programming?
- Are there other, more modern possibilities?



Object model vs. relational model

- Relational model is very seldom directly compatible with the data model of any programming language.
- When an object oriented language is used integration may be really challenging.
- Term *impedance mismatch* is used for this

Impedance mismatch

Mismatch	Description
Granularity	Sometimes you will have an object model which has more classes than the number of corresponding tables in the database.
Inheritance	RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.
Identity	A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (<code>a==b</code>) and object equality (<code>a.equals(b)</code>).
Associations	Object-oriented languages represent associations using object references where as an RDBMS represents an association as a foreign key column.
Navigation	The ways you access objects in Java and in a RDBMS are fundamentally different.

Challenges in integration

Object model

- SW engineering
- Class as the basic concept
- Objects (instances) formed according to the model defined in class
- Data and operations (methods) tightly bundled
- Associations, inheritance
- Each class has its own operations to change the data
- Language specific data types

Relational model

- Mathematics (set theory)
- Table (relation) schema as the basic concept
- Rows follow the structure defined in table schema
- Only data is modeled, no methods.
- References
- Fixed, unicersal DML (typically SQL) for changing the data
- SQL data types (system dependent variation)



Solutions

- Object-Relational Mapping (ORM) is the solution to handle all the above impedance mismatches and challenges.
- ORM can be coded by hand from the scratch
- DAO design pattern can be used to organize the application (database operations are isolated in their own classes.
- ORM can be done automatically by using ready-made ORM frameworks e.g. [Hibernate](#)



Advantages of ORM

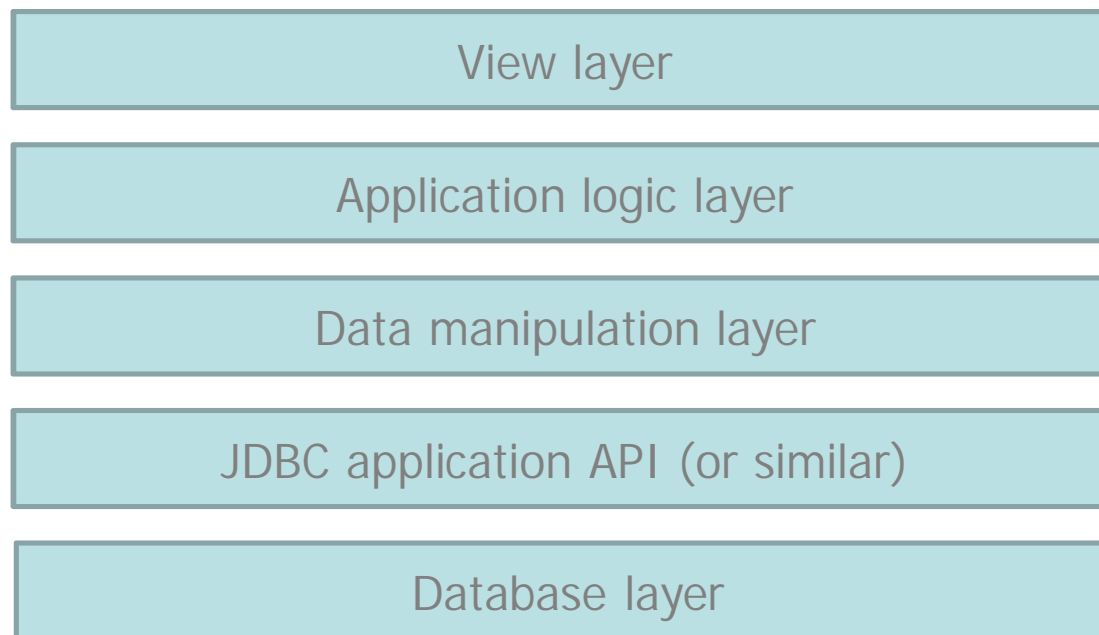
Advantages of ORM automation over plain JDBC and DAO

1. Lets business code access objects rather than DB tables
2. Hides details of SQL queries from OO logic
3. Based on JDBC 'under the hood'
4. No need to deal with the database implementation
5. Entities based on business concepts rather than database structure
6. Transaction management and automatic key generation
7. Fast development of application

DAO Design Pattern

- The application's entity classes and classes responsible for database functionality are isolated from each other
- Data manipulation layer is needed in the application architecture (DAL – Data Access Layer)
- Options for DAL implementation:
 - One single class that is responsible for all the database operations
 - For each entity class its own DAO class with just one instance
 - For each entity class its own DAO class with an instance for every object

DAO Design Pattern



Lähde Silander, Ollikainen, Peltomäki: Java

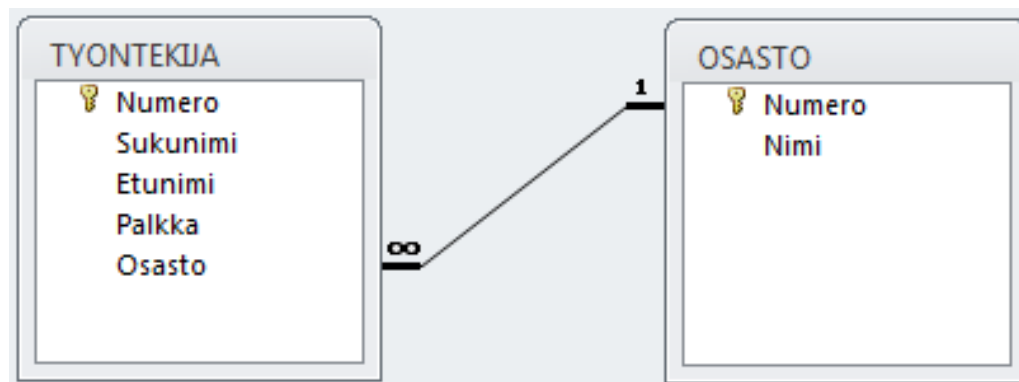


Hibernate

- Hibernate is an open source library for ORM automatization in Java environment
- The original idea is to describe the relationship between classes and database tables with XML – nowadays also annotations can be used for the purpose
- Hibernate automatically translates method calls in object environment to operations of relational database
- Even database creation and forming the database connection can be made automatic

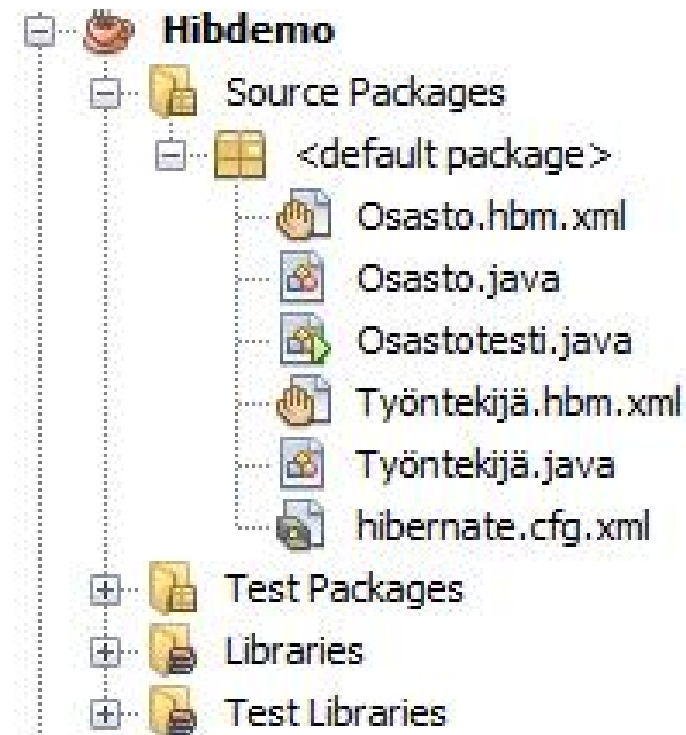
Example

- Let us use Hibernate in a sample situation where
 - The Java application has two classes: **Työntekijä** (**Employee**) and **Osasto** (**Department**)
 - The relational database has corresponding tables: **TYONTEKIJA**, **OSASTO**



Files that are needed

- In this case we need six files:
 - Class files
Työntekijä.java and **Osasto.java**
 - Configuration file
hibernate.cfg.xml
 - Mapping files for each class
Työntekijä.hbm.xml and **osasto.hbm.xml**
 - Class **Osastotesti.java** is a test class that contains on testiluokka, joka sisältää **main** method.



Class files

- Entity classes are plain Java classes tavallisia Java-luokkia.
- Use of Hibernate is not visible in them (compare to annotations)
- You have to write
 - An empty constructor with no parameters
 - Set and get methods for each attribute that is handled by Hibernate

```
public class Osasto {  
  
    private int numero;  
    private String nimi;  
  
    public Osasto(int numero, String nimi) {  
        this.numero = numero;  
        this.nimi = nimi;  
    }  
  
    public Osasto() {  
    }  
  
    public String getNimi() {  
        return nimi;  
    }  
  
    public int getNumero() {  
        return numero;  
    }  
  
    public void setNimi(String nimi) {  
        this.nimi = nimi;  
    }  
  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
  
}
```


Class files

```
public class Työntekijä {  
  
    private int numero;  
    private String sukunimi;  
    private String etunimi;  
    private double palkka;  
    private Osasto osasto;  
  
    public Työntekijä() {  
    }  
  
    public void korotaPalkkaa(double korotusprosentti) {  
        palkka += palkka * korotusprosentti;  
    }  
  
    public Työntekijä(int numero, String sukunimi,  
        String etunimi, double palkka, Osasto osasto) {  
        this.numero = numero;  
        this.sukunimi = sukunimi;  
        this.etunimi = etunimi;  
        this.palkka = palkka;  
        this.osasto = osasto;  
    }  
}
```

```
    public String getEtunimi() {  
        return etunimi;  
    }  
  
    public void setEtunimi(String etunimi) {  
        this.etunimi = etunimi;  
    }  
  
    public int getNumero() {  
        return numero;  
    }  
  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
  
    public Osasto getOsasto() {  
        return osasto;  
    }  
  
    public void setOsasto(Osasto osasto) {  
        this.osasto = osasto;  
    }  
  
    public double getPalkka() {  
        return palkka;  
    }  
  
    public void setPalkka(double palkka) {  
        this.palkka = palkka;  
    }  
  
    public String getSukunimi() {  
        return sukunimi;  
    }  
  
    public void setSukunimi(String sukunimi) {  
        this.sukunimi = sukunimi;  
    }  
}
```

Configuration file hibernate.cfg.xml

- In the configuration file you specify i.a. database driver, server connection, username and password
- Value **create** of **hbm2ddl.auto** arvo **create** indicates that the tables will be created every time , when the program is run
 - Other options: **validate**, **update**, **create-drop**.
- **mapping** elements refer to class specific mapping files
- To save space xml specification and identification of the document type have been omitted

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLInnoDBDialect
    </property>
    <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
    </property>
    <property name="hibernate.connection.url">
      jdbc:mysql://mysql.metropolia.fi:3306/olliv
    </property>
    <property name="hibernate.connection.username">
      olliv
    </property>
    <property name="hibernate.connection.password">
      [redacted]
    </property>
    <property name="hbm2ddl.auto">create</property>
    <mapping resource="Osasto.hbm.xml"/>
    <mapping resource="Työntekijä.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Class specific mapping files

- A mapping file is written for each class; it describes how the attributes are mapped to database columns
- Each **Osasto** object is represented as a row in **osasto** table
- Field **osastonumero** is the primary key, which corresponds to attribute **numero** in the Java class
- There is also a field **nimi** in the database table. It is stored in a 40 character long string. It corresponds to an attribute with the same name in the Java class.

```
<hibernate-mapping>
  <class name="Osasto" table="osasto">
    <id name="numero" type="integer">
      <column name="osastonumero"/>
    </id>
    <property name="nimi" type="string">
      <column length="40" name="nimi"/>
    </property>
  </class>
</hibernate-mapping>
```

Class specific mapping files

- There is a one-to-many relationship between tables **tyontekija** and **osasto**
- It is mapped with the help of **many-to-one** element
- The table structure (**CREATE TABLE** statements) are built automatically based on the class specific mapping information

```
<hibernate-mapping>
  <class name="Työntekijä" table="tyontekija">
    <id name="numero" type="integer">
      <column name="numero"/>
    </id>
    <property name="sukunimi" type="string">
      <column length="40" name="sukunimi"/>
    </property>
    <property name="etunimi" type="string">
      <column length="40" name="etunimi"/>
    </property>
    <property name="palkka" type="double">
      <column name="palkka"/>
    </property>
    <many-to-one name="osasto"
      column="osastonumero"
      not-null="true"/>
  </class>
</hibernate-mapping>
```

Session and writing data

- First a Hibernate session is created
- The session object is asked to start a transaction
- Operations are written within the session (**saveOrUpdate** method)
- At last the transaction is committed
- N.B. The following import statements have to be added in the beginning of a class that uses hibernate:

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

```
public class Osastotesti {

    public static void main(String[] args) {

        SessionFactory istuntotehdas = new Configuration().configure().
            buildSessionFactory();
        Session istunto = istuntotehdas.openSession();

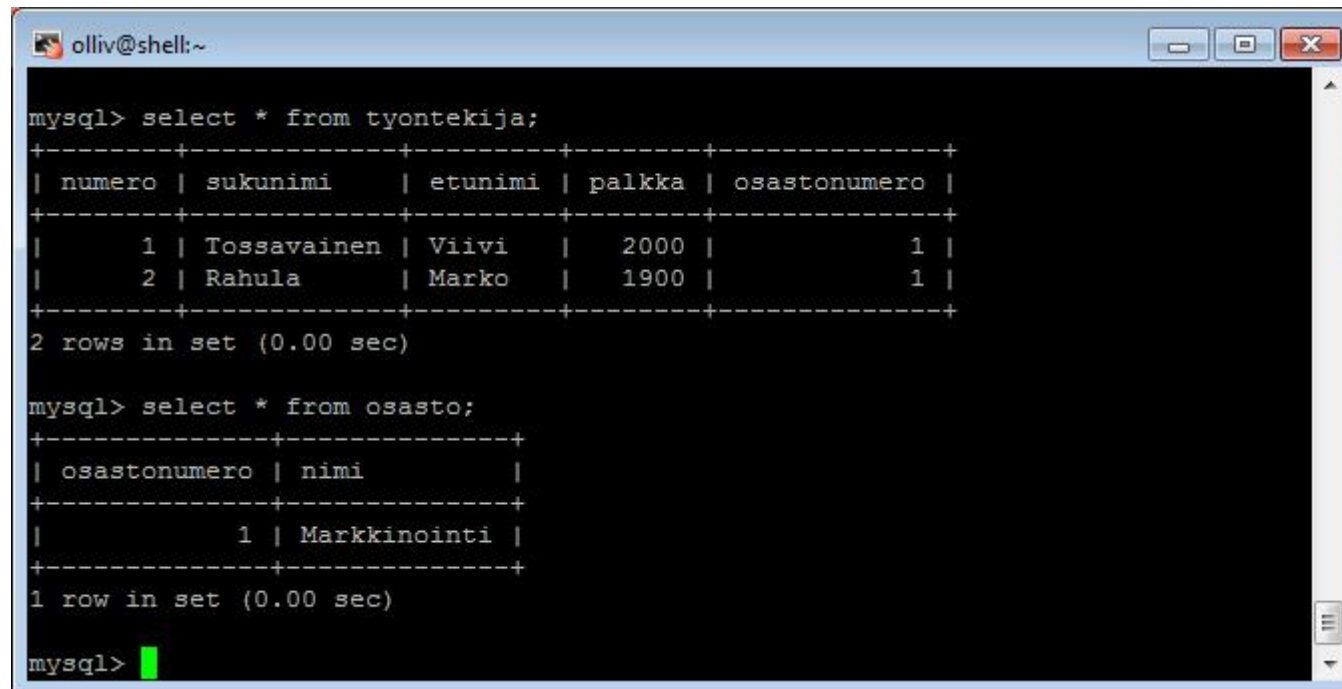
        Transaction transaktio = null;

        try {
            transaktio = istunto.beginTransaction();
            Osasto os1 = new Osasto(1, "Markkinointi");
            Työntekijä tt1 = new Työntekijä (1, "Tossavainen",
                "Viivi", 2000, os1);
            Työntekijä tt2 = new Työntekijä(2, "Rahula",
                "Marko", 1900, os1);
            istunto.saveOrUpdate(os1);
            istunto.saveOrUpdate(tt1);
            istunto.saveOrUpdate(tt2);

            transaktio.commit();
        } catch (Exception e) {
            if (transaktio != null & transaktio.isActive()) {
                try {
                    transaktio.rollback();
                } catch (HibernateException e1) {
                    System.err.println("Tapahtuman " +
                        "peruutus epäonnistui.");
                }
            }
            e.printStackTrace();
        } finally {
            istunto.close();
        }
    }
}
```

Changes in the database

- The changes made in the Java program are automatically updated in the database:



A terminal window titled 'olliv@shell:~' showing MySQL commands and their output. The first command is 'mysql> select * from tyontekija;', which returns a table with 5 columns: numero, sukunimi, etunimi, palkka, and osastonumero. It shows two rows of employee data. The second command is 'mysql> select * from osasto;', which returns a table with 2 columns: osastonumero and nimi, showing one row for the department 'Markkinointi'.

```
olliv@shell:~  
mysql> select * from tyontekija;  
+-----+-----+-----+-----+-----+  
| numero | sukunimi   | etunimi | palkka | osastonumero |  
+-----+-----+-----+-----+-----+  
|      1 | Tossavainen | Viivi   | 2000   | 1            |  
|      2 | Rahula     | Marko   | 1900   | 1            |  
+-----+-----+-----+-----+-----+  
2 rows in set (0.00 sec)  
  
mysql> select * from osasto;  
+-----+-----+  
| osastonumero | nimi          |  
+-----+-----+  
|            1 | Markkinointi |  
+-----+-----+  
1 row in set (0.00 sec)  
  
mysql> █
```

Reading data

```
Työntekijä tt3 = new Työntekijä();  
istunto.load(tt3, 7);  
System.out.println(tt3.getSukunimi());
```

- The rows above fetch the data for employee number 7 based on the value of the primary key (**load** method)
- The data is automatically placed in the instance variables of object **tt3**
- Data can also be fetched with the help of queries in SQL or HQL (*Hibernate Query Language*)