Hibernate

Bringing Java and SQL closer together

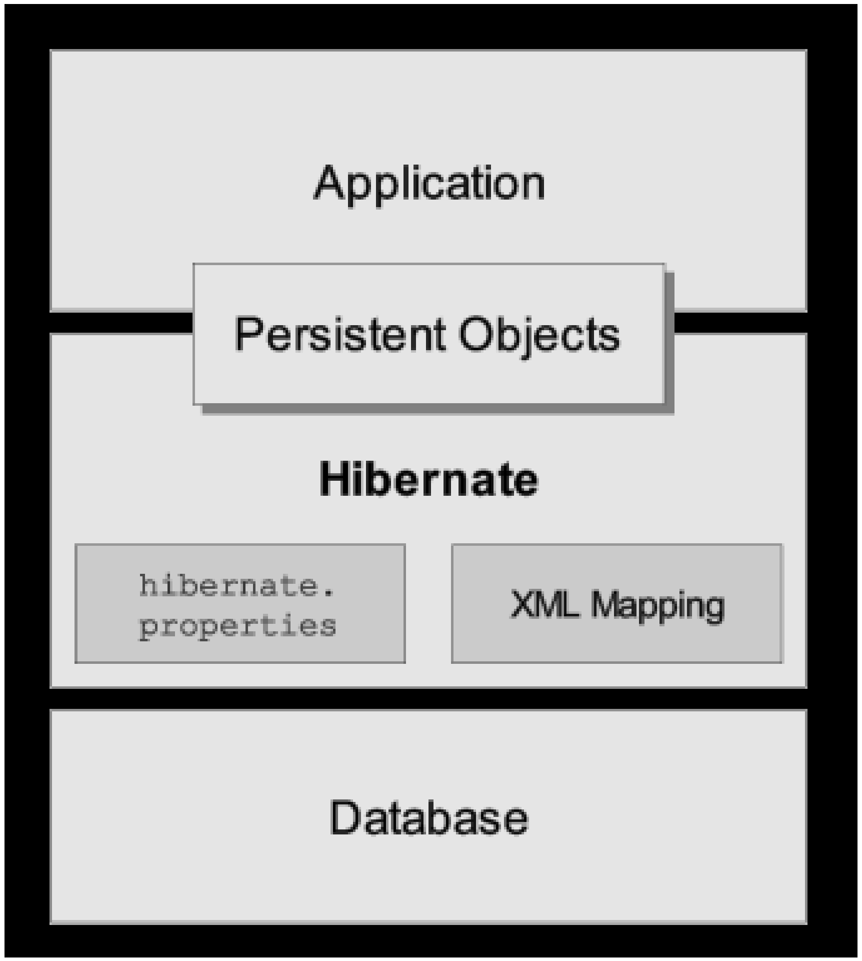
EHSDI
e B u z i m a

Hibernate

- An object-relational mapping (ORM) library for Java
- Allows us to interact with a database using regular Java objects
- Provides a closer relationship between your Java objects and tables in your database

Overview

- Runs on top of JDBC
- Allows us to easily *persist* objects in a database
- Works with different database types, and so is an *abstraction* layer



Object mapping

```
class Patient {
  private int id;
  private String name;
  private Date dob;

  …
}
```

Hibernate allows to associate database columns with class properties

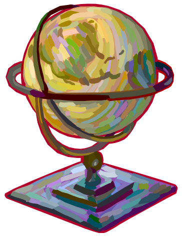
id	name	dob
1	Ben	1993-04-21
2	Rowan	1981-05-28
3	Rita	1983-01-07

EHSDI
eBuzima

Object mapping

- We tell Hibernate how to map a Java class to a database table
- Hibernate can then
  - Save instances as new table rows
  - Load instances of that class from rows in the table
  - Update instances

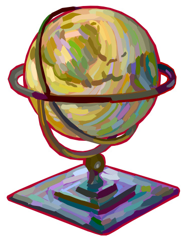
Mapping files



- We describe the relationship between a Java class and a database table in an XML file, e.g.

```
<hibernate-mapping package="eh203.emr">

    <class name="Patient" table="patients">
        <id name="id" column="patient_id">
            <generator class="native"/>
        </id>
        <property name="dob" type="date" column="dob"/>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

Mapping files



Java class name

Package of Java class

Table name

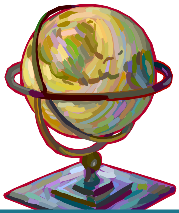
```
<hibernate-mapping   ackage="eh203.emr">

    <class name="Patient" table="patients">
        <id name="id" column="patient_id">
            <generator class="native"/>
        </id>
        <property name="dob" type="date" column="dob"/>
        <property name="name"/>
    </class>

</hibernate-      ing>
```

Class property name

Column type

Table column name

Mapping files

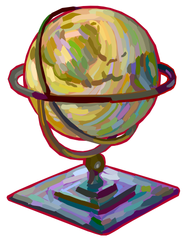


Types are not Java or SQL types but special Hibernate mapping types

```
<hibernate-mapping package="eh203.emr">

    <class name="Patient" table="patients"
        <id name="id" column="patient_id">
            <generator class="native"/>
        </id>
        <property name="dob" type="date" column="dob"/>
        <property name="name"/>
    </class>

</hib
```

When column name is not specified it defaults to the property name

Mapping types



- Often Hibernate can guess the correct type using reflection
- But sometimes we need to state the type explicitly, e.g.
  - Should a property of type `java.util.Date` map to a column of type TIMESTAMP, DATETIME or DATE?

Hibernate types (basic)

Hibernate mapping type	Java	SQL
integer, long, short, float, double, character, boolean	Primitive types, e.g. `int`	`INT`, `BIGINT` etc
string	`java.lang.String`	`VARCHAR`
text	`java.lang.String`	`TEXT`, `CLOB`
binary	`byte[]`	`BINARY`

See http://docs.jboss.org/hibernate/stable/core/reference/en/html/mapping.html#mapping-types

EHSDI
eBuzima

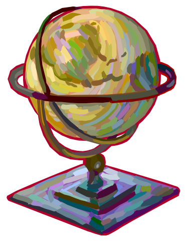
Hibernate types (dates)

- These require some extra attention…

Hibernate mapping type	Java	SQL
date	`java.util.Date`	DATE
time	`java.util.Date`	TIME
timestamp	`java.util.Date`	TIMESTAMP
calendar	`java.util.Calendar`	TIMESTAMP
calendar_date	`java.util.Calendar`	DATE

See http://docs.jboss.org/hibernate/stable/core/reference/en/html/mapping.html#mapping-types

Ids and keys



- Primary keys in the database have to be linked to "id" properties on the class, e.g.

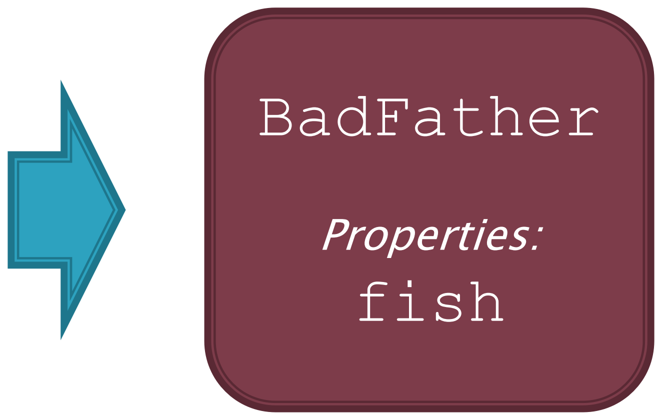
```
<hibernate-mapping package="eh203.emr">

    <class name="Patient" table="patients">
        <id name="id" column="patient_id">
            <generator class="native"/>
        </id>
        <property name="dob" type="    etime" column="dob"/>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

Tells Hibernate to use the id value generated by the database (AUTO_INCREMENT)

Bean properties

- The mapping files reference the names of the properties of the class as a Java Bean
- They are NOT the names of the fields, e.g.

```
class BadFather {
  private int snake;
  public int getFish() {
    return snake;
  }
}
```



XML configuration

- For example…

Database connection settings

```
<?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="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
        <property name="connection.username">dbuser</property>
        <property name="connection.password">dbpass</property>
        <property name="connection.pool_size">1</property>

        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <property name="current_session_context_class">thread</property>
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>

        <mapping resource="pharmacy/domain/Patient.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Mapping files

Configuration

- Different values for **hbm2ddl.auto** tell Hibernate what do to the database schema when starting up...
  - *validate*: validate the schema, makes no changes to the database.
  - *update*: update the schema.
  - *create*: creates the schema, destroying previous data.
  - *create-drop*: drop the schema at the end of the session.

Configuration

- Setting the value of **show_sql** to *true* allows us to see all of the SQL which Hibernate executes
- We can use Hibernate with different types of SQL server by changing the value of **dialect** – to use MySQL we set it to…

  org.hibernate.dialect.MySQLDialect

EHSDI eBuzima

Project structure

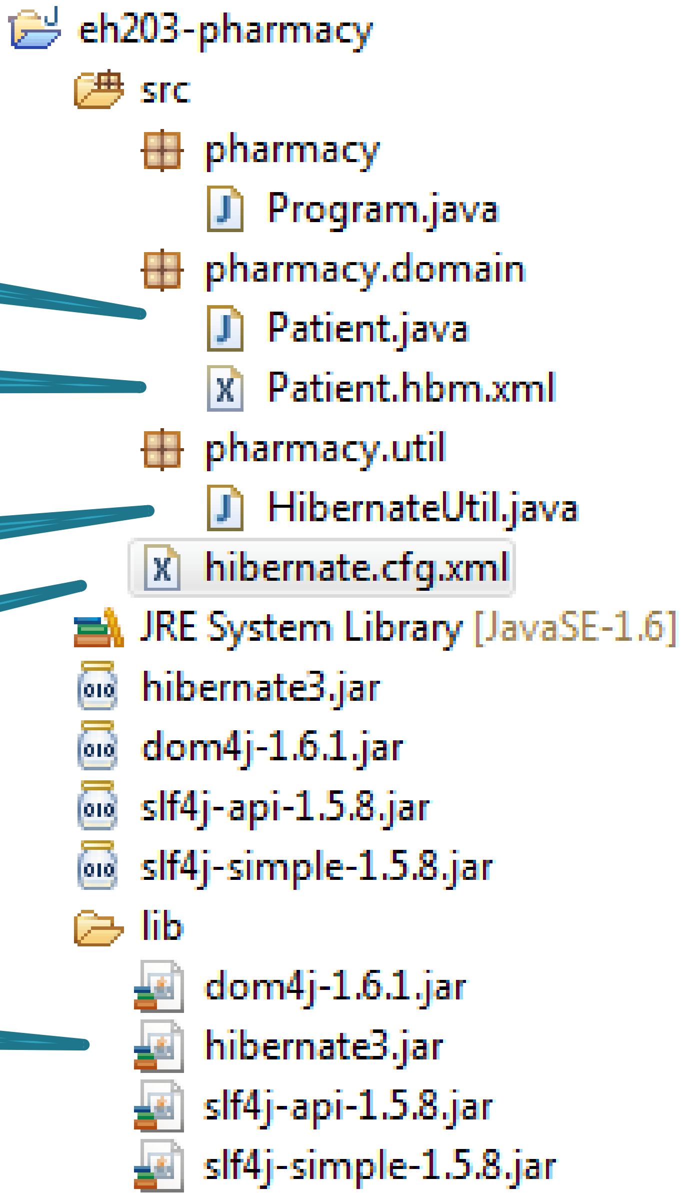
Java bean classes

Corresponding mapping XML files

Standard utility class to create Hibernate sessions

Hibernate configuration file

Hibernate JAR and other required JARS



```
eh203-pharmacy
    src
        pharmacy
            Program.java
        pharmacy.domain
            Patient.java
            Patient.hbm.xml
        pharmacy.util
            HibernateUtil.java
        hibernate.cfg.xml
    JRE System Library [JavaSE-1.6]
    hibernate3.jar
    dom4j-1.6.1.jar
    slf4j-api-1.5.8.jar
    slf4j-simple-1.5.8.jar
    lib
        dom4j-1.6.1.jar
        hibernate3.jar
        slf4j-api-1.5.8.jar
        slf4j-simple-1.5.8.jar
```

EHSDI
eBuzima

Creating domain objects: classes

```java
public class Patient {
  protected int patientId;
  protected String name;
  protected Date dob;

  public Patient() {
  }

  public Patient(String name, Date dob) {
    this.name = name;
    this.dob = dob;
  }

  public int getPatientId() {
    return patientId;
  }
  ...
}
```

Hibernate requires classes with default constructors

We can also define an explicit constructor for our use

All properties are accessed through BEAN methods

HibernateUtil

- This is a utility class that we can put in our Hibernate projects
- Why isn't it part of the Hibernate library??? Who knows…
- Simply allows to get a valid Hibernate session object anytime we need one, e.g.

```
Session session
   = HibernateUtil.getSessionFactory().getCurrentSession();
```

EHSDI eBuzima

Persisting objects

- We can use the Hibernate session object to persist an instance of a mapped class, e.g.

```
Session session
 = HibernateUtil.getSessionFactory().getCurrentSession();

Patient patient = new Patient("Bob", new Date());

session.beginTransaction();
session.save(patient);
session.getTransaction().commit();
```

Creates a new row in the database

EHSDI
eBuzima

Persisting objects

- We can make changes to persisted object and then tell Hibernate to update the database, e.g.

```
Patient patient = new Patient("Bob", new Date());

session.beginTransaction();

session.save(patient);

patient.setName("Bob Jones");

session.save(patient);

session.getTransaction().commit();
```

Creates a new row in the database

Updates the existing row in the database

EHSDI
eBuzima

Persisting objects

- Hibernate looks at the id property of an object to know if it is already in the database, e.g.

```
Patient patient = new Patient("Bob", new Date());

session.beginTransaction();

session.save(patient);

patient.setName("Bob Jones");

session.save(patient);

session.getTransaction().commit();
```

Patient id is 0

Now id is > 0

Hibernate knows to UPDATE rather than CREATE because id > 0