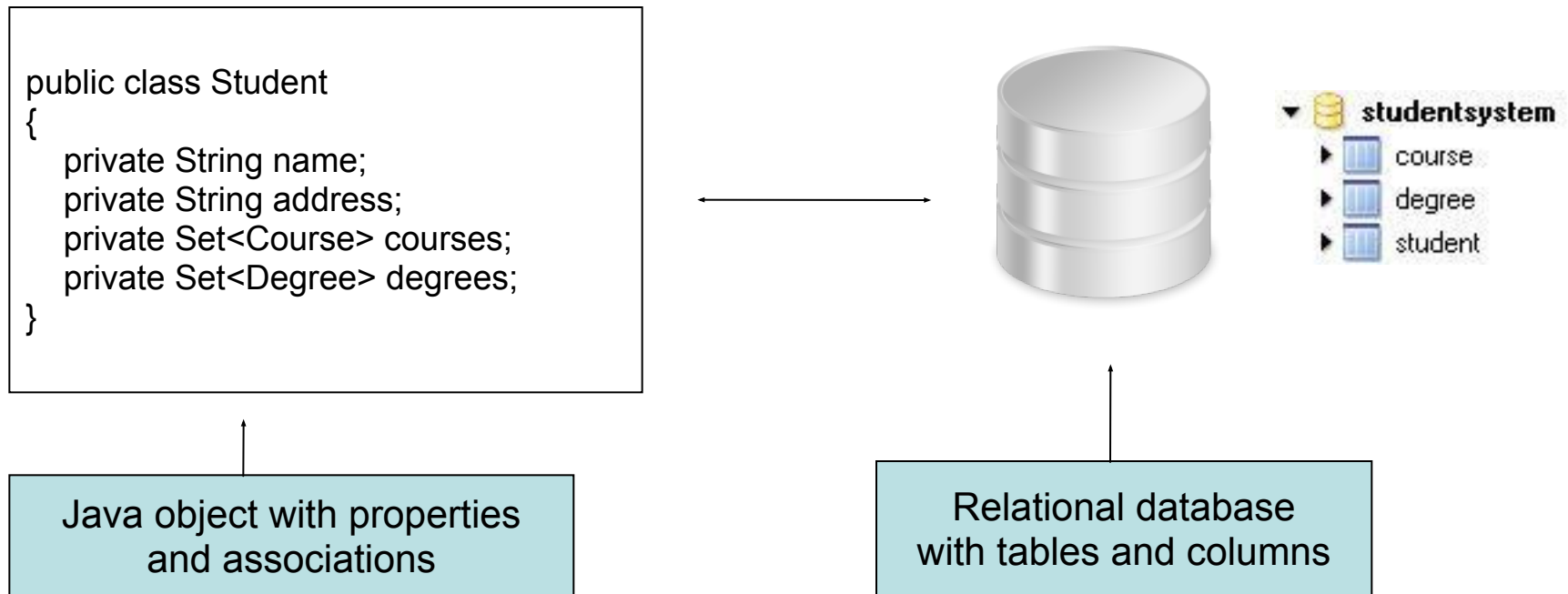# Object <u>Relational</u> Mapping (ORM) and Hibernate

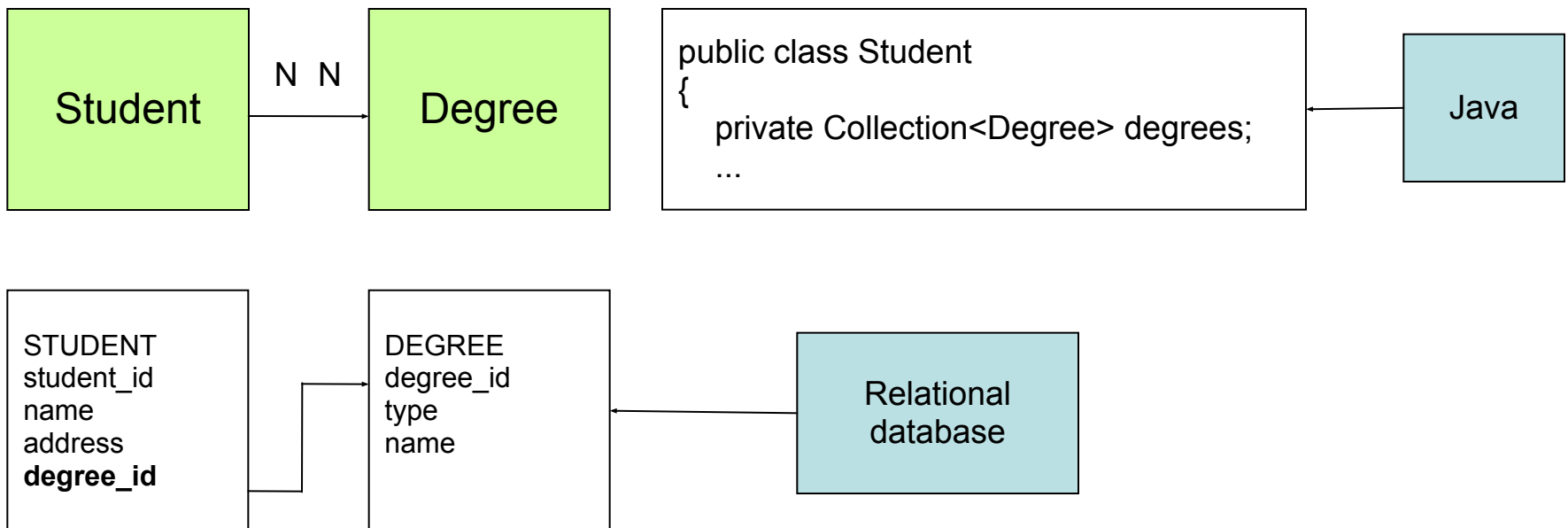**INF5750/9750 - Lecture 2 (Part III)**

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?

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

studentsystem
- course
- degree
- student

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

```
public class Student
{
    private Collection<Degree> degrees;
    ...
```

Student N N Degree

Java

STUDENT
student_id
name
address
degree_id

DEGREE
degree_id
type
name

Relational database

Need for 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

```
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
}
```
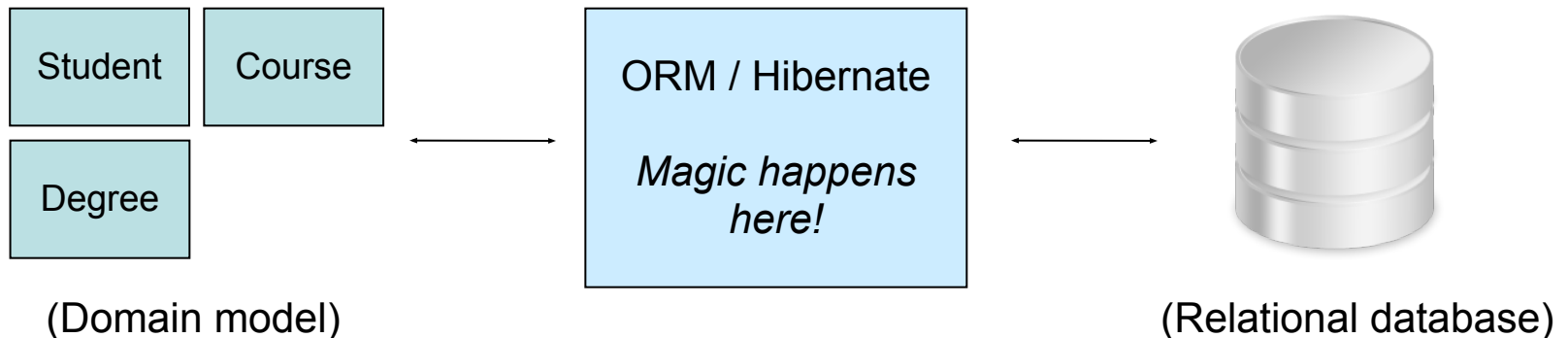
Student   Course

Degree

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
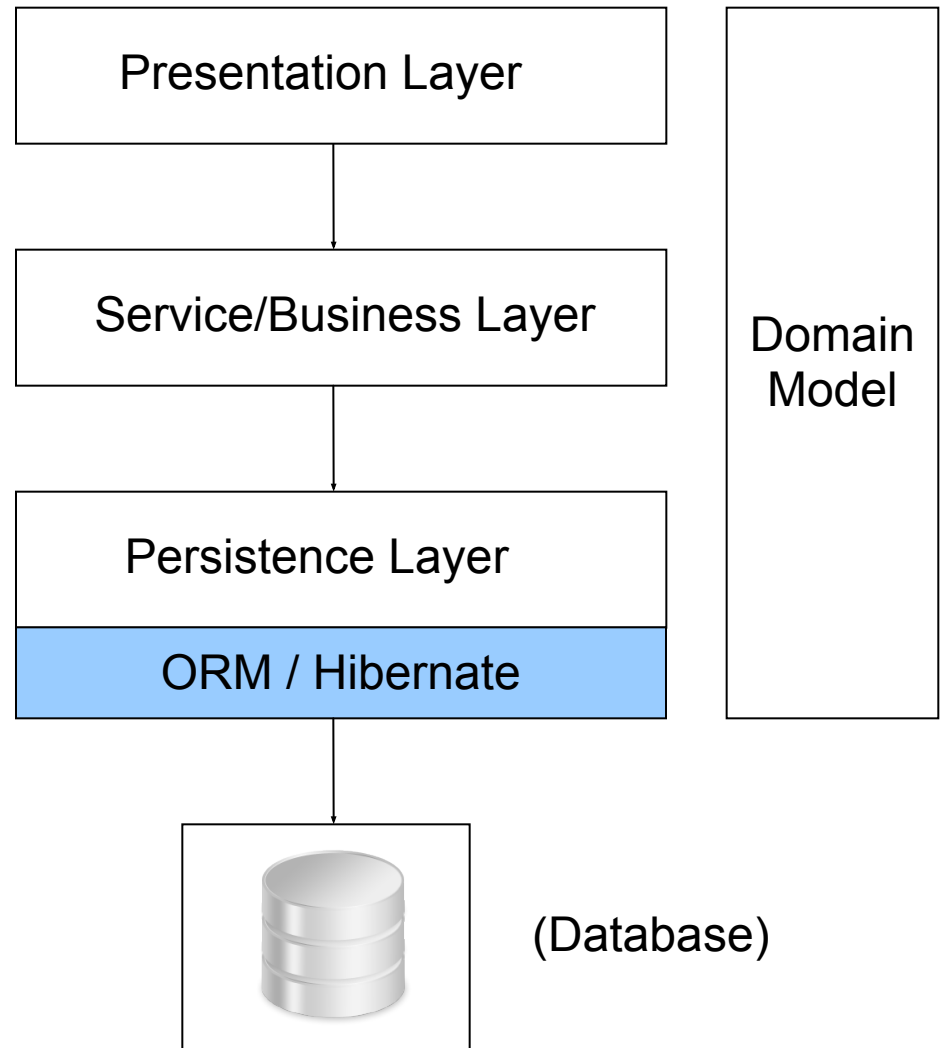 - Lacks necessary features

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
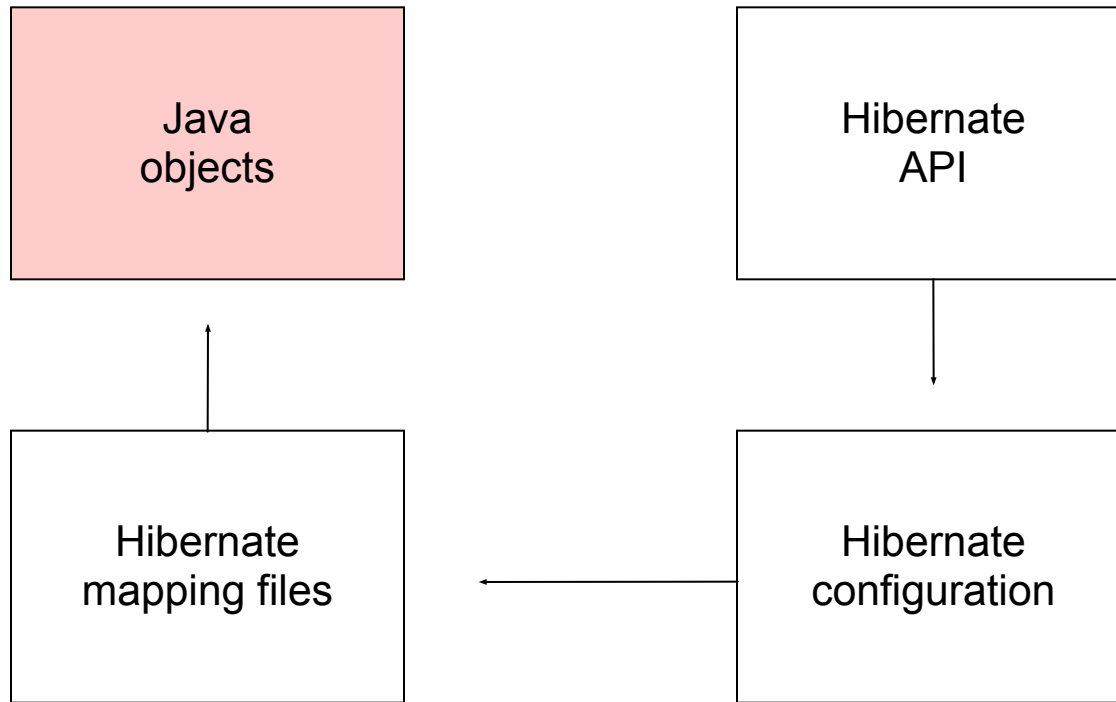- *Transparent*: Your object model is unaware

Student	Course

Degree

ORM / Hibernate

Magic happens here!

(Domain model)

(Relational database)

ORM and Architecture

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

```
┌─────────────────────────────┐     ┌──────────┐
│     Presentation Layer      │     │          │
└─────────────────────────────┘     │          │
             │                      │          │
             ▼                      │          │
┌─────────────────────────────┐     │          │
│    Service/Business Layer    │     │  Domain  │
└─────────────────────────────┘     │  Model   │
             │                      │          │
             ▼                      │          │
┌─────────────────────────────┐     │          │
│      Persistence Layer       │     │          │
├─────────────────────────────┤     │          │
│       ORM / Hibernate        │     │          │
└─────────────────────────────┘     └──────────┘
             │
             ▼
       ┌──────────┐
       │    🛢     │  (Database)
       └──────────┘
```

Example app: The EventManager

```
┌─────────────────┐          ┌─────────────────┐
│                 │          │                 │
│      Java       │          │    Hibernate    │
│     objects     │          │       API       │
│                 │          │                 │
└─────────────────┘          └─────────────────┘
         ▲                            │
         │                            ▼
┌─────────────────┐          ┌─────────────────┐
│                 │          │                 │
│    Hibernate    │ ◄─────── │    Hibernate    │
│  mapping files  │          │  configuration  │
│                 │          │                 │
└─────────────────┘          └─────────────────┘
```

Java objects (POJO)

Identifier property
(optional)

No-argument constructor
(required)

Declare accessors and mutators for persistent fields (optional)

- Properties need *not* be declared public - Hibernate can persist a property with a default, protected or private get / set pair.

Prefer non-final classes
(optional)

```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
}
```
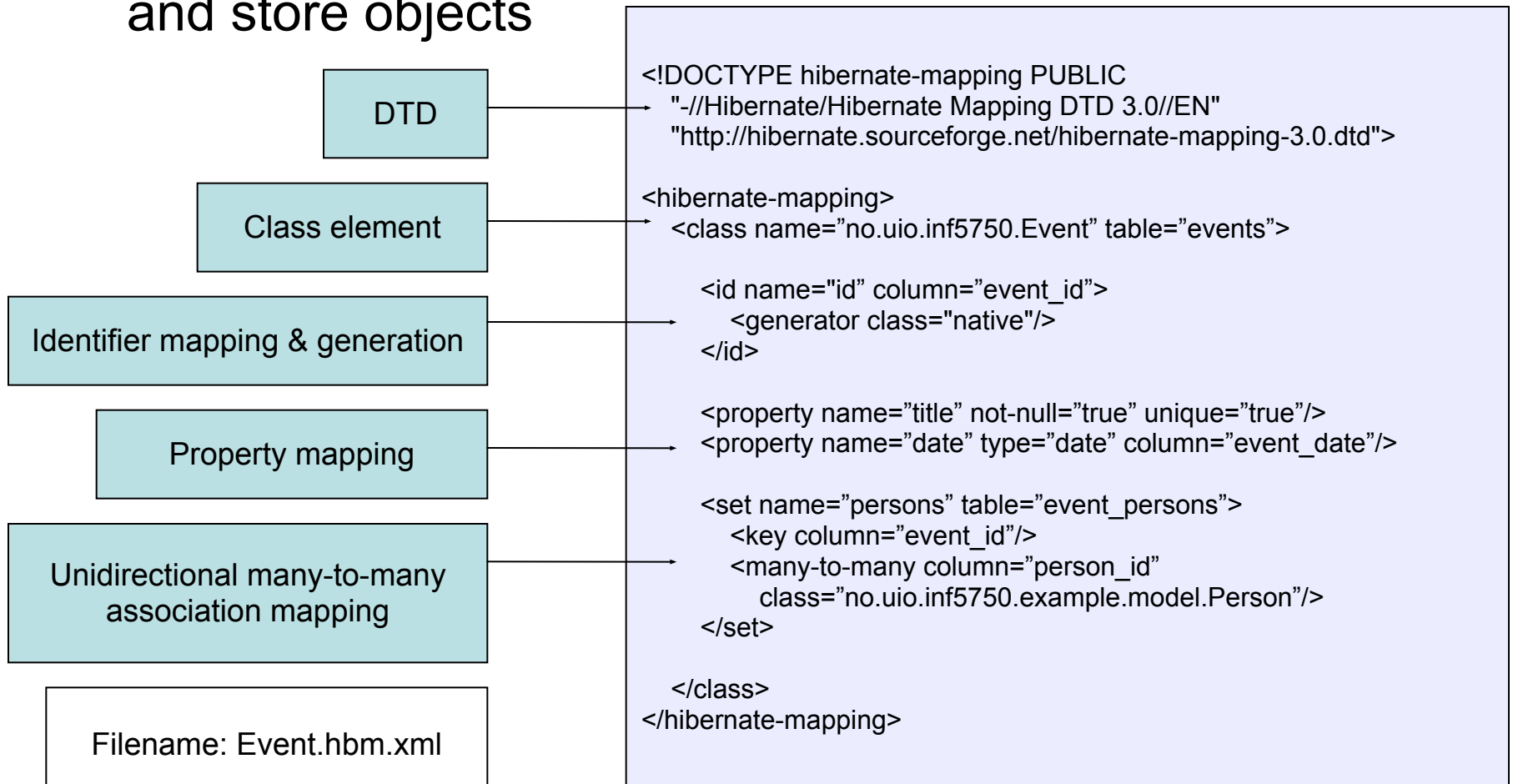
Example app: The EventManager

Java objects	Hibernate API

Hibernate mapping files	Hibernate configuration

Hibernate mapping files

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

DTD

Class element

Identifier mapping & generation

Property mapping

Unidirectional many-to-many association mapping

Filename: Event.hbm.xml

```xml
<!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>
```
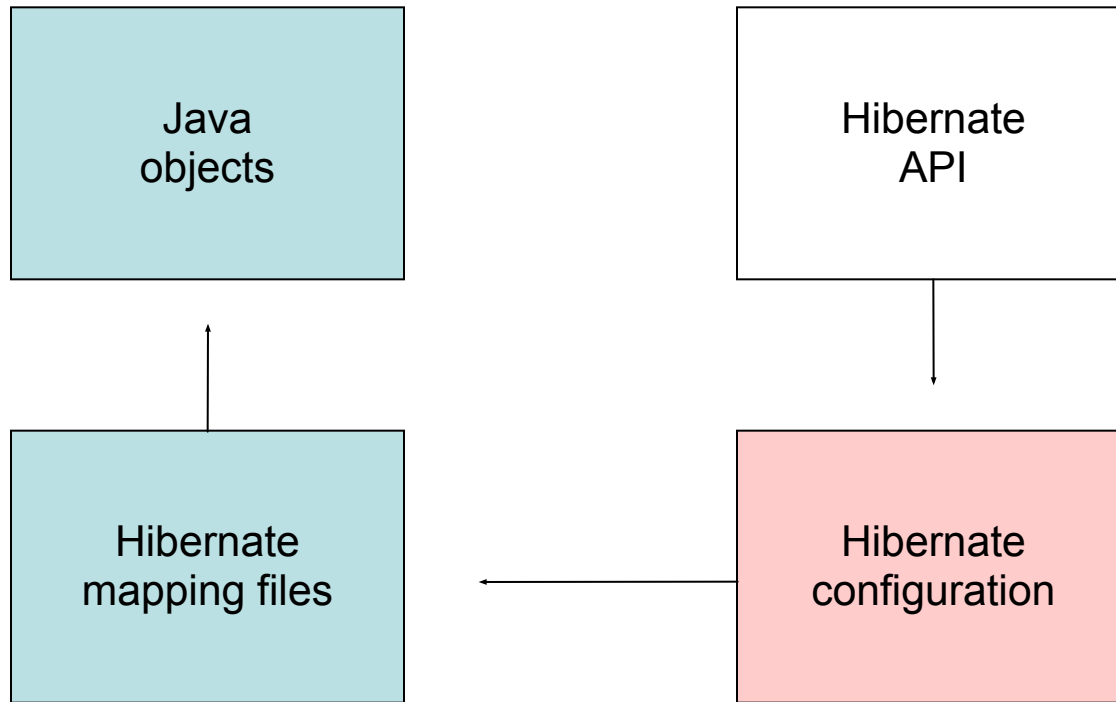
Property mapping

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

Association mapping

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

```
...
<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

Hibernate mapping types

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

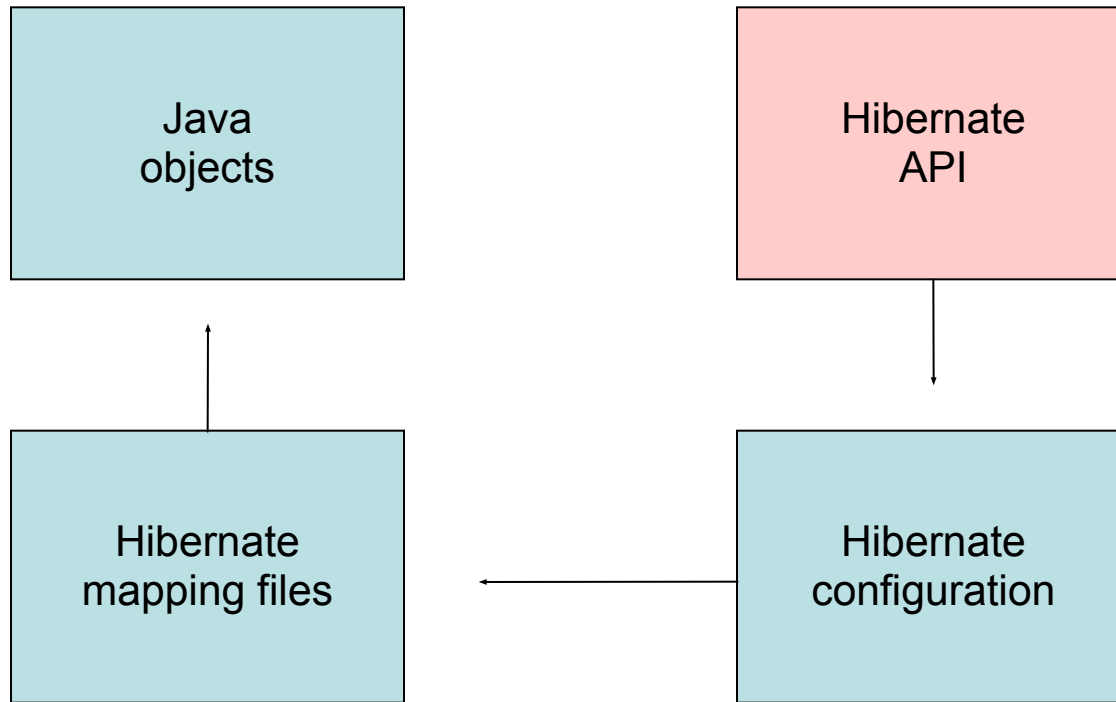| Java type	Hibernate type	SQL type
java.lang.String	string	VARCHAR
java.util.Date	date, time	DATE, TIME
java.lang.Integer, int	integer	INT
java.lang.Class	class	varchar
java.io.Serializable	serializable	BLOB, BINARY

Example app: The EventManager

Hibernate configuration

- Also referred to hibernate properties
- 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

The SessionFactory interface

- When all mappings have been parsed by the *org. hibernate.cfg.Configuration*, the application must obtain a factory to get *org.hibernate.Session*
- SessionFactory provides *org.hibernate.Session* instances to the application
- Shared among application threads
- Most important method is *getCurrentSession*

```
SessionFactory sessionFactory = cfg.buildSessionFactory();
```

OR let *Spring* inject it for you

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
     <property name="mappingResources">
    ….
</bean>
```

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	`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
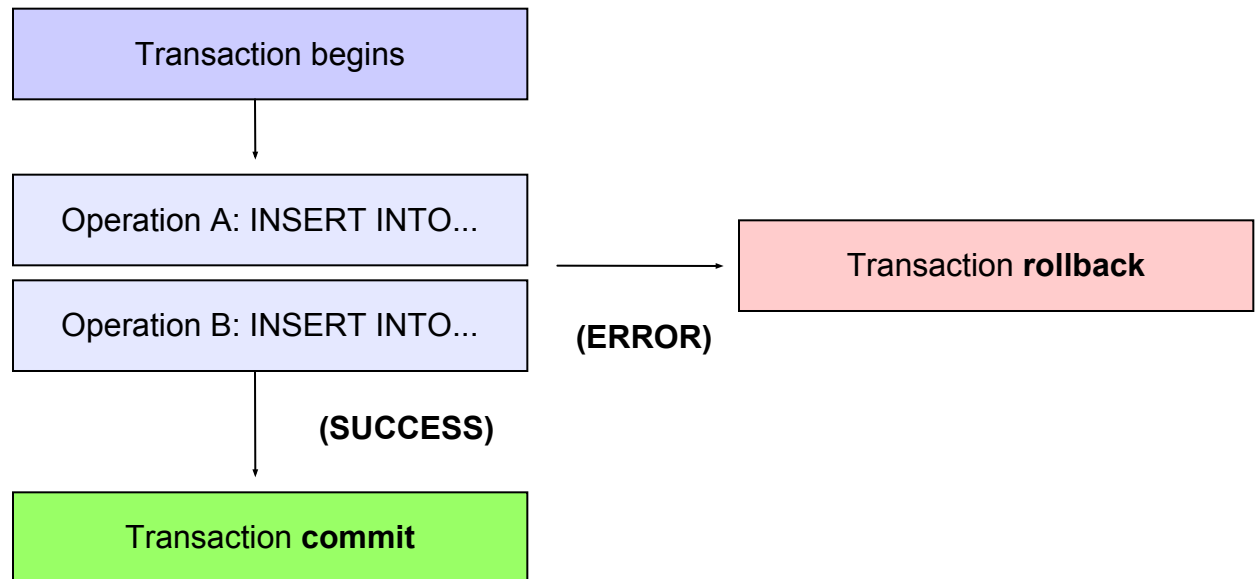- Criteria used for *programmatic* query creation

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!

Transaction begins

Operation A: INSERT INTO...

Operation B: INSERT INTO...

Transaction **rollback**

(ERROR)

(SUCCESS)

Transaction **commit**
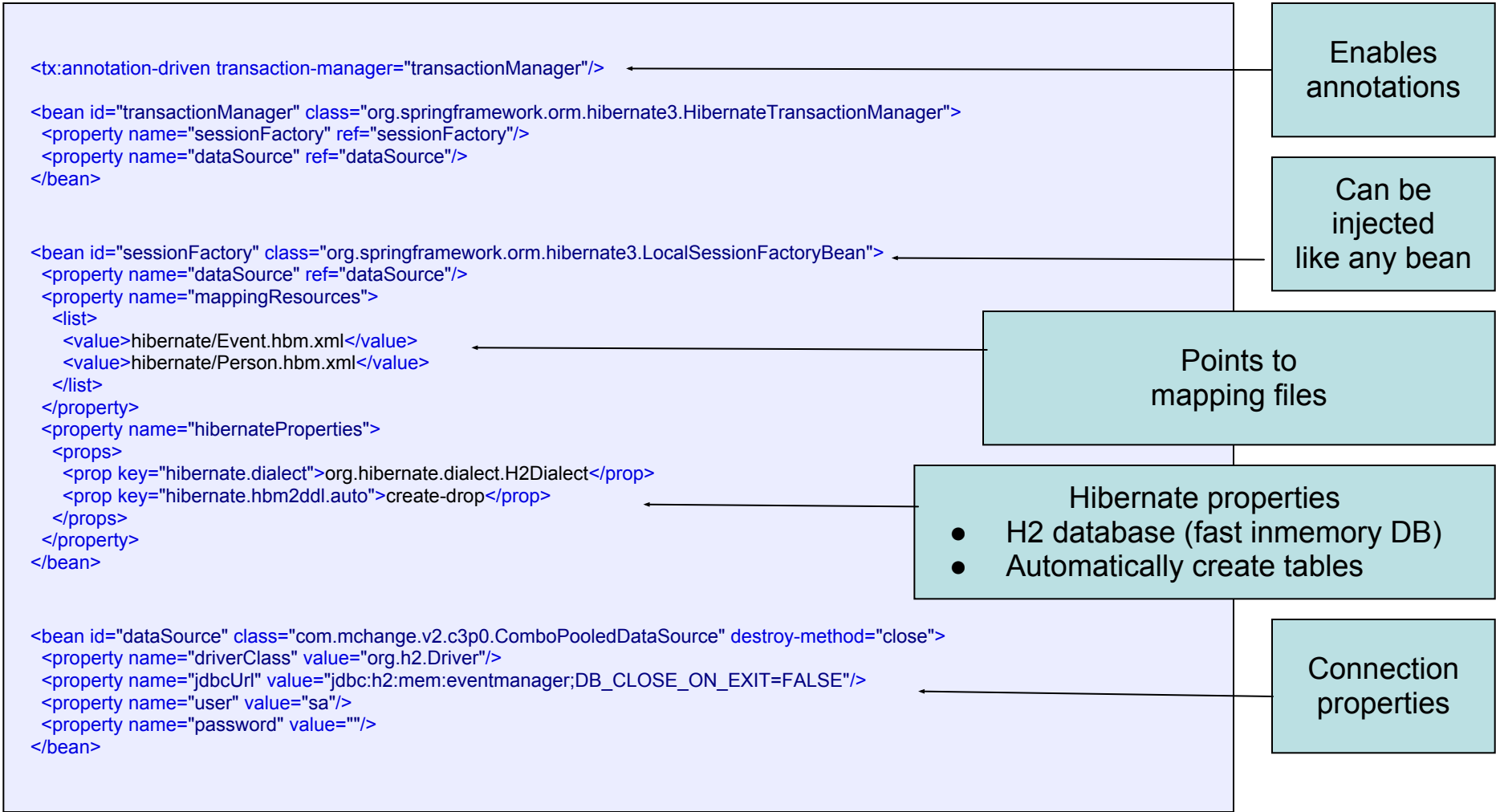
Hibernate in real-life apps

- 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*

- Hibernate's own connection pooling is basic. Hibernate allows external JDBC pooling through library called C3P0

Spring-Hibernate dependencies

```xml
...
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency>
```
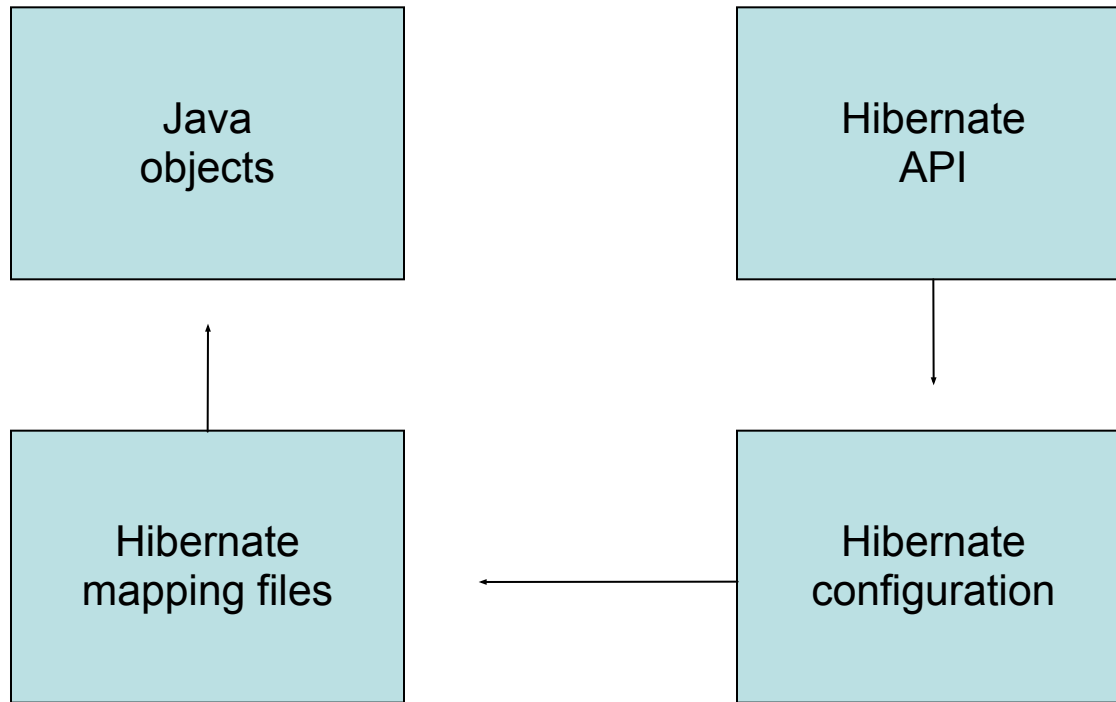
```xml
...
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>${hibernate.version}</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>3.5.1-Final</version>
</dependency>
<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
- H2 database (fast inmemory DB)
- Automatically create tables

Connection properties

Example: The EventManager

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
- Database vendor independence
  - The underlying database is abstracted away
  - Can be configured outside the application

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

Resources

- Spring documentation chapter 13

http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/orm.html

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