

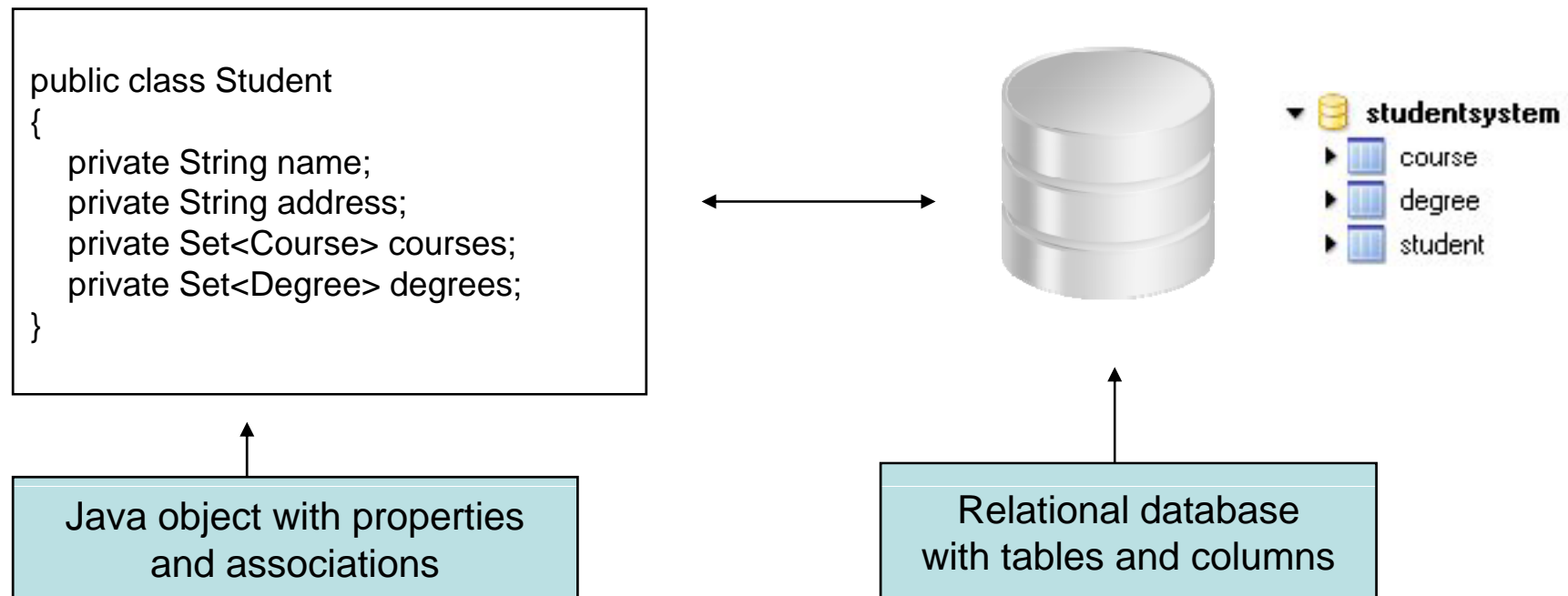
Object-Relational Mapping (ORM)

and

Hibernate

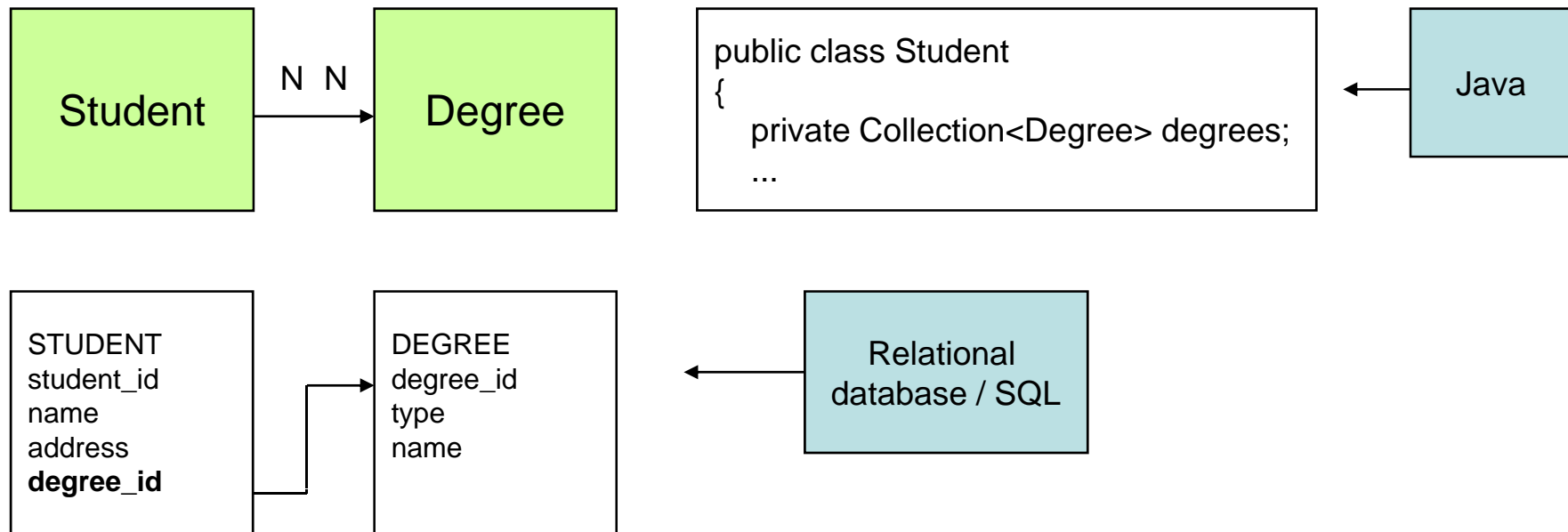
Problem area

- 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?



Problem area

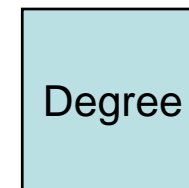
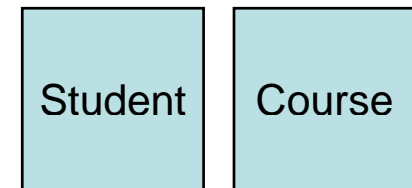
- How to map associations between objects?
 - References are directional, foreign keys not
 - Foreign keys can't represent many-to-many associations



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!

(Domain model)



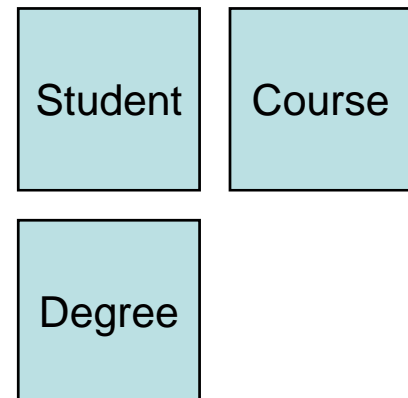
(Database)

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

```
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
}
```

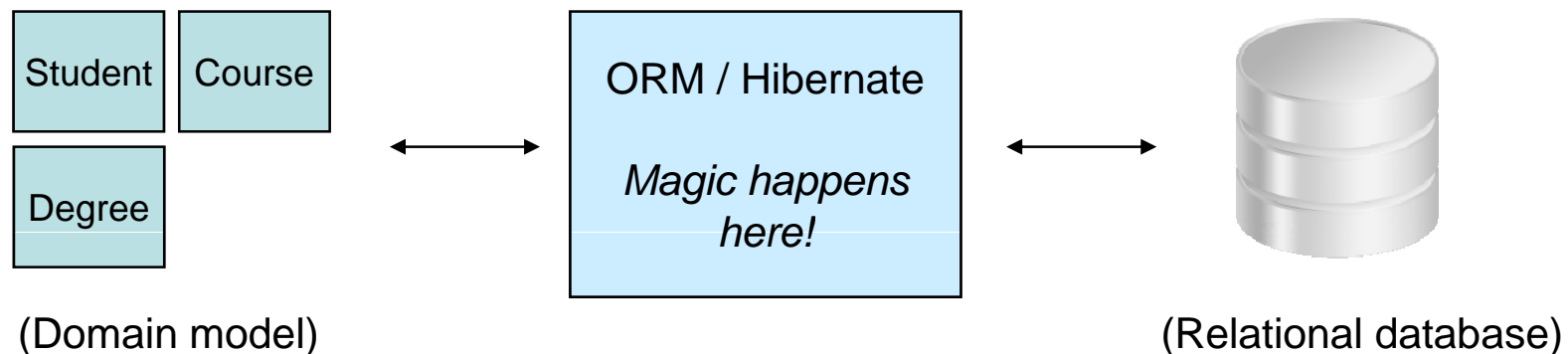


Approaches to ORM

- 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

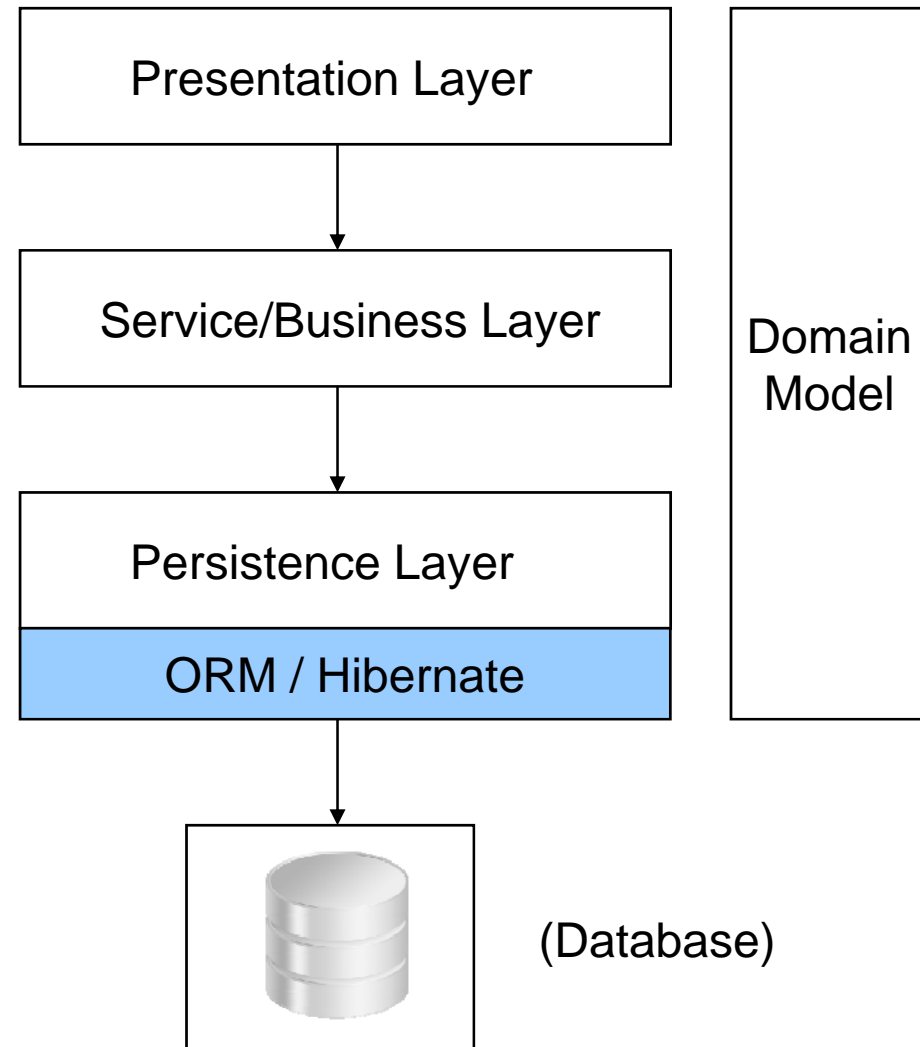
The preferred solution

- 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

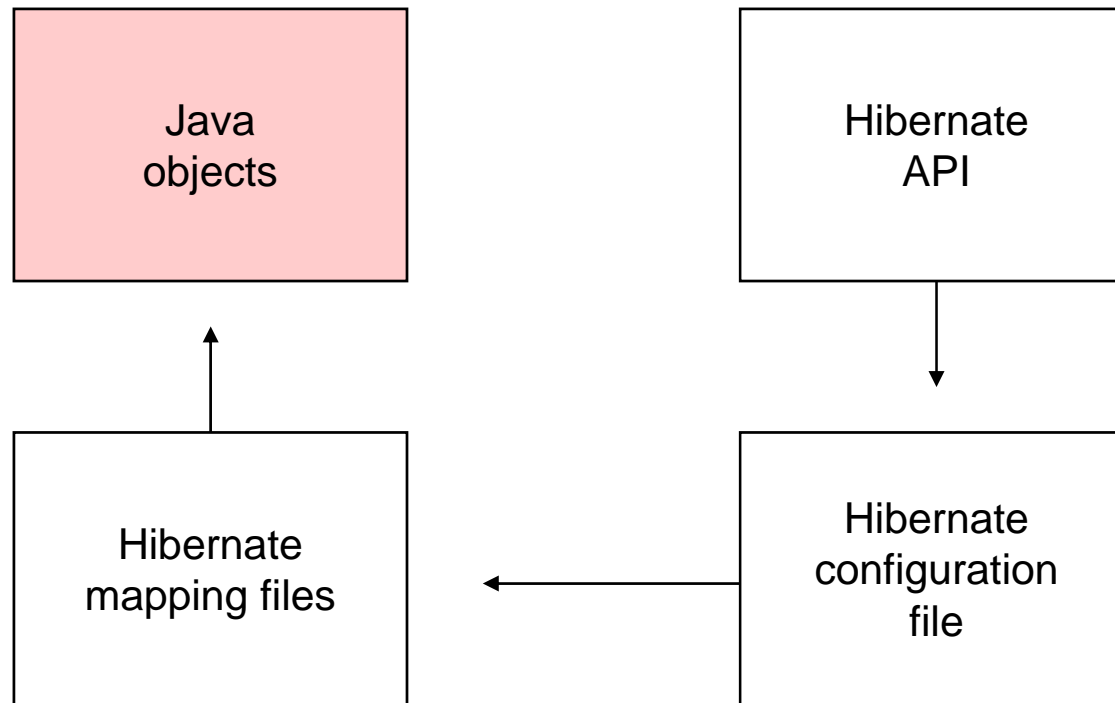


ORM and Architecture

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



Example app: The EventManager



Java objects

Identifier property



No-argument constructor



Follows the
JavaBean naming
conventions



```
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;
    }

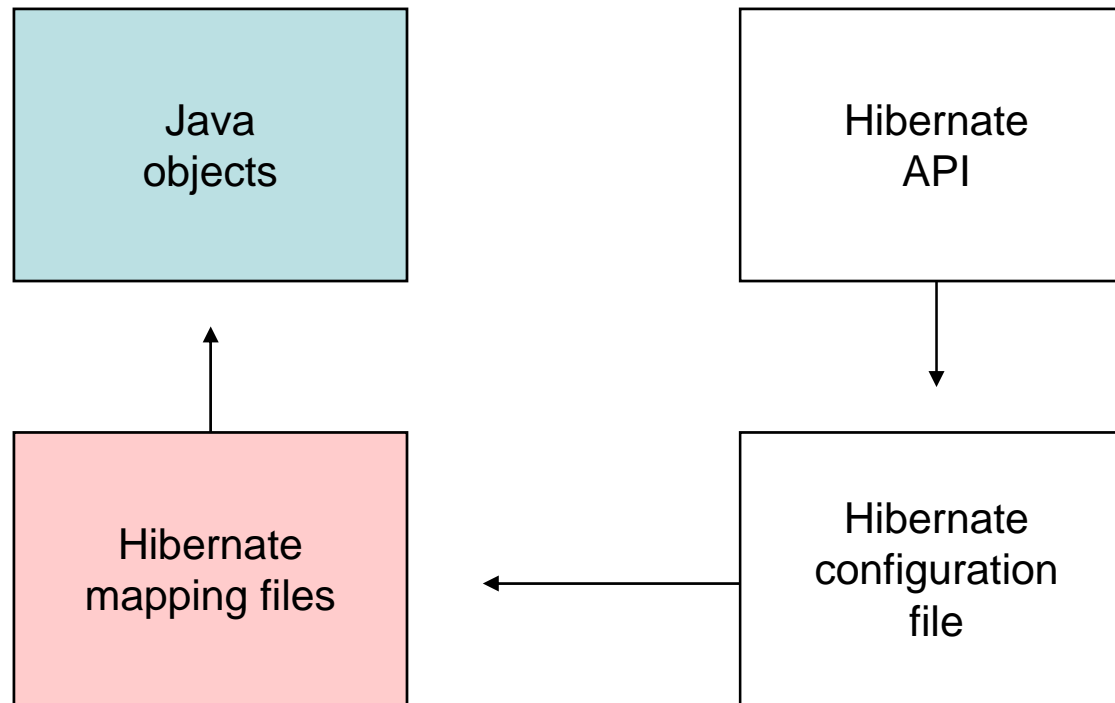
    private void setId( int id ) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

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

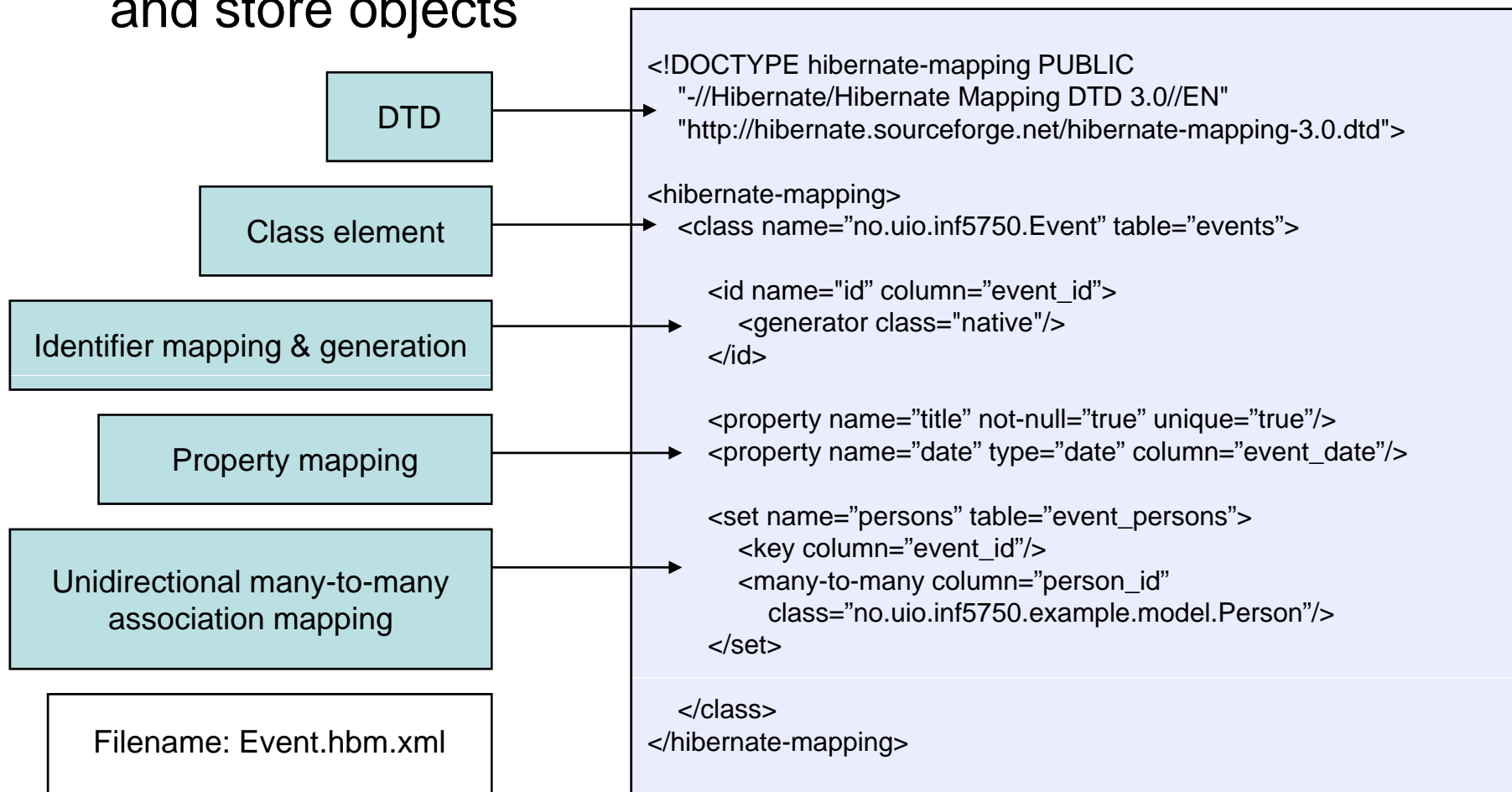
    // Getter and setter for date and persons
}
```

Example app: The EventManager

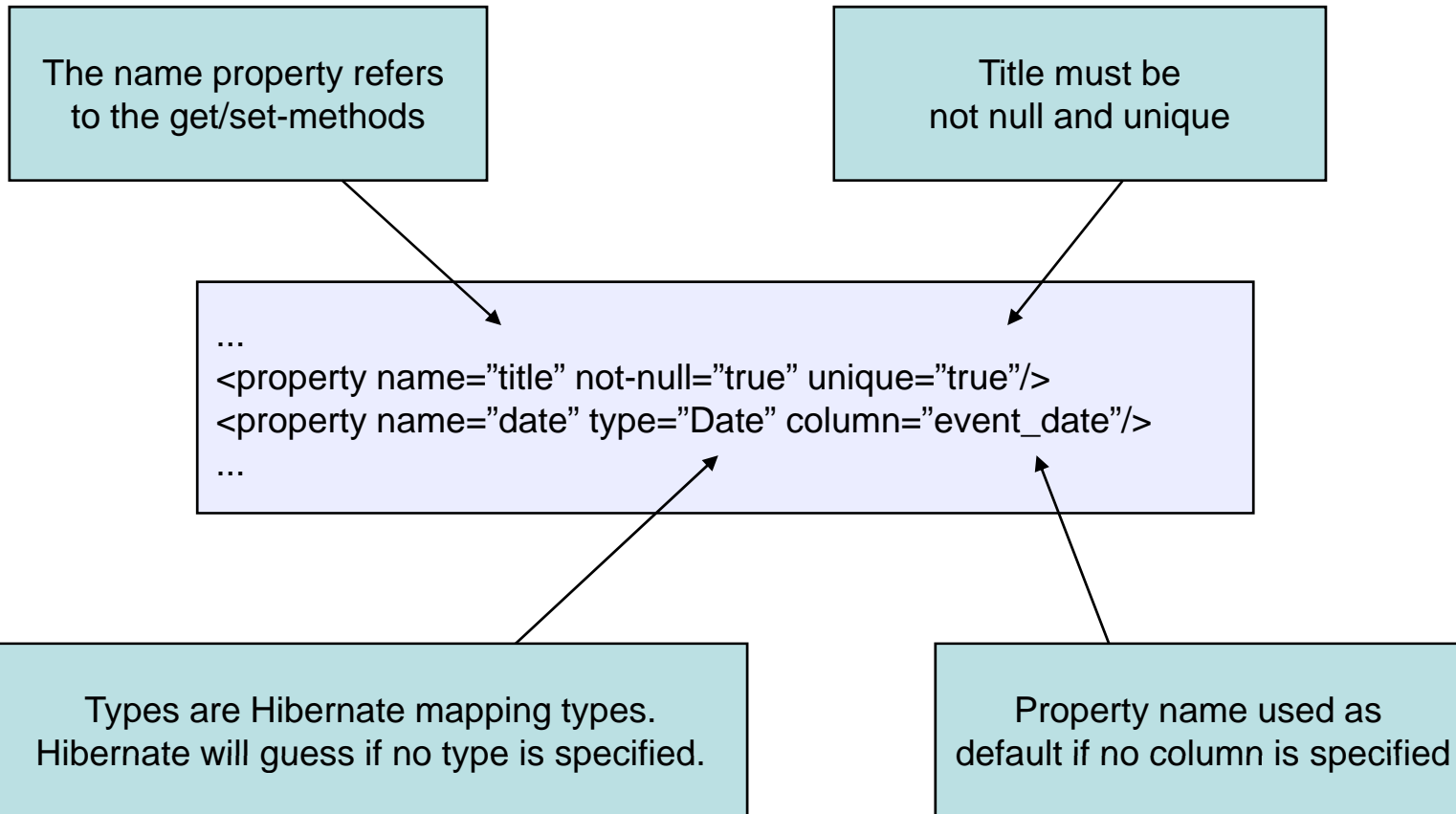


Hibernate mapping files

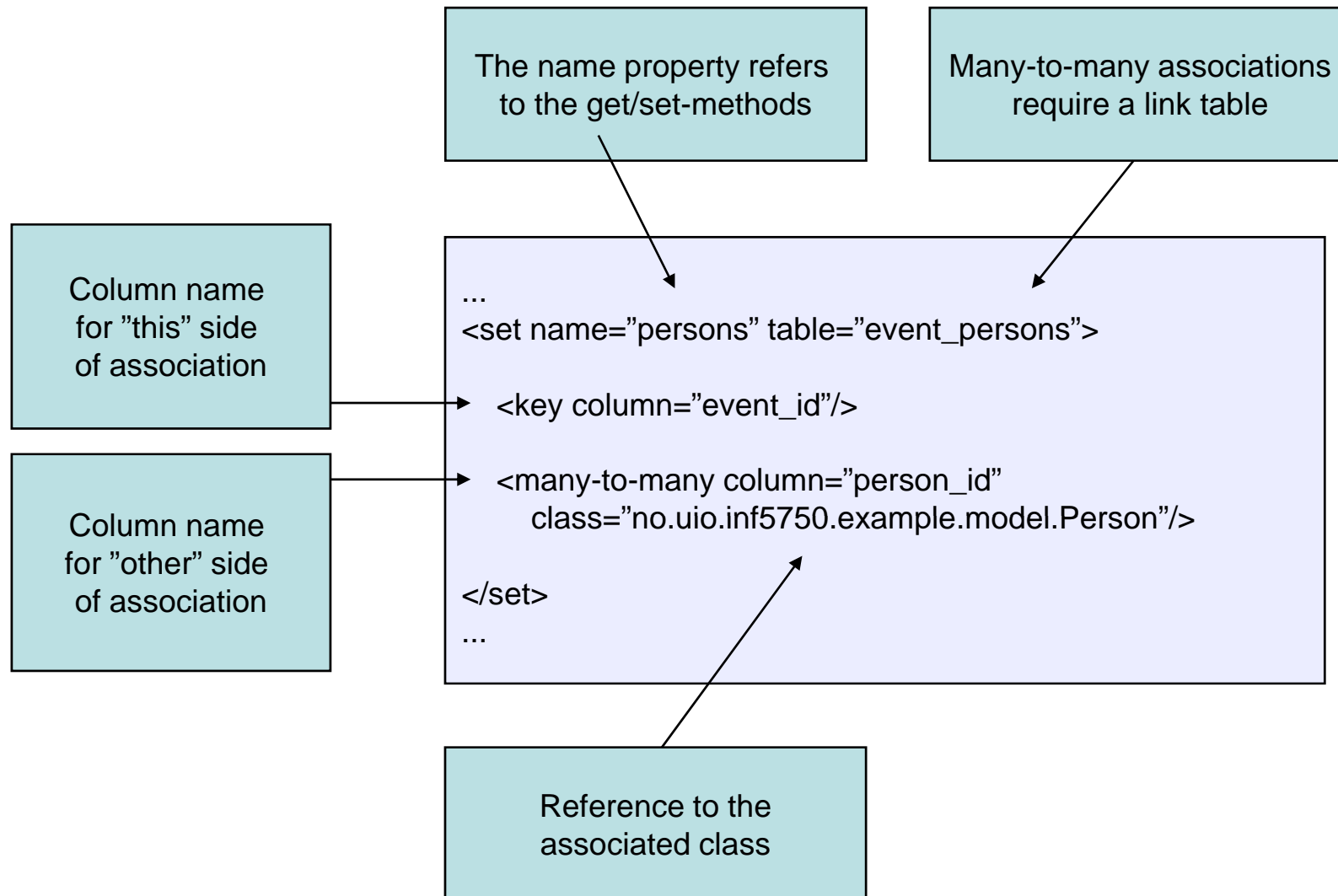
- Tells Hibernate which tables and columns to use to load and store objects



Property mapping



Association mapping

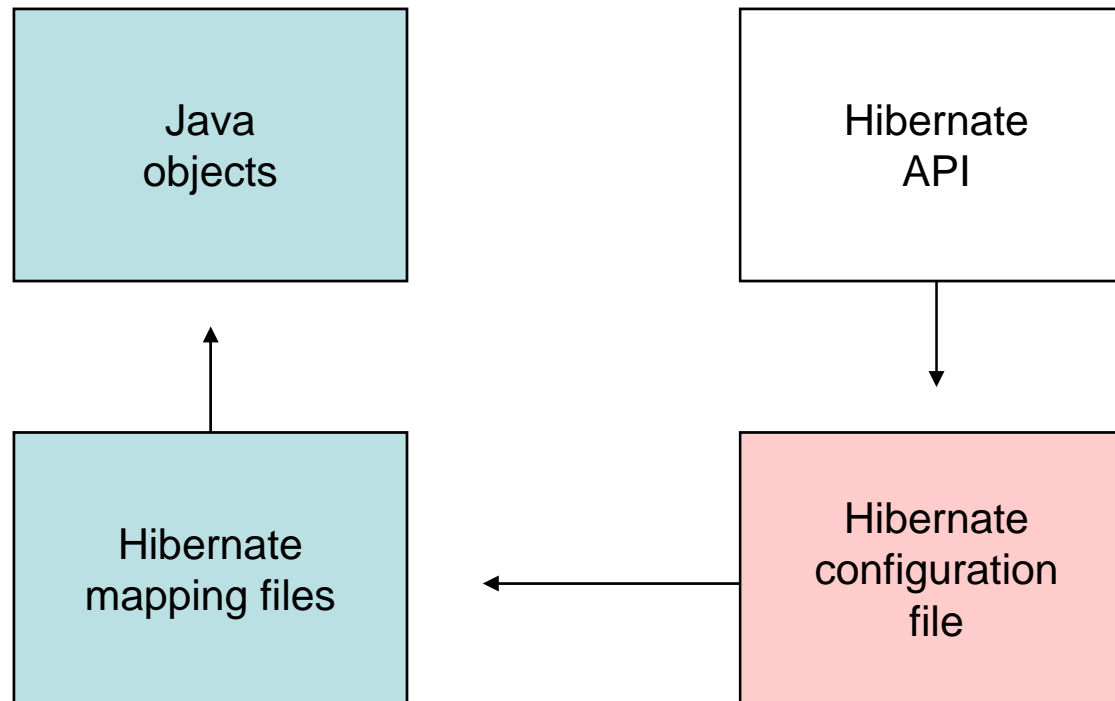


Hibernate mapping types

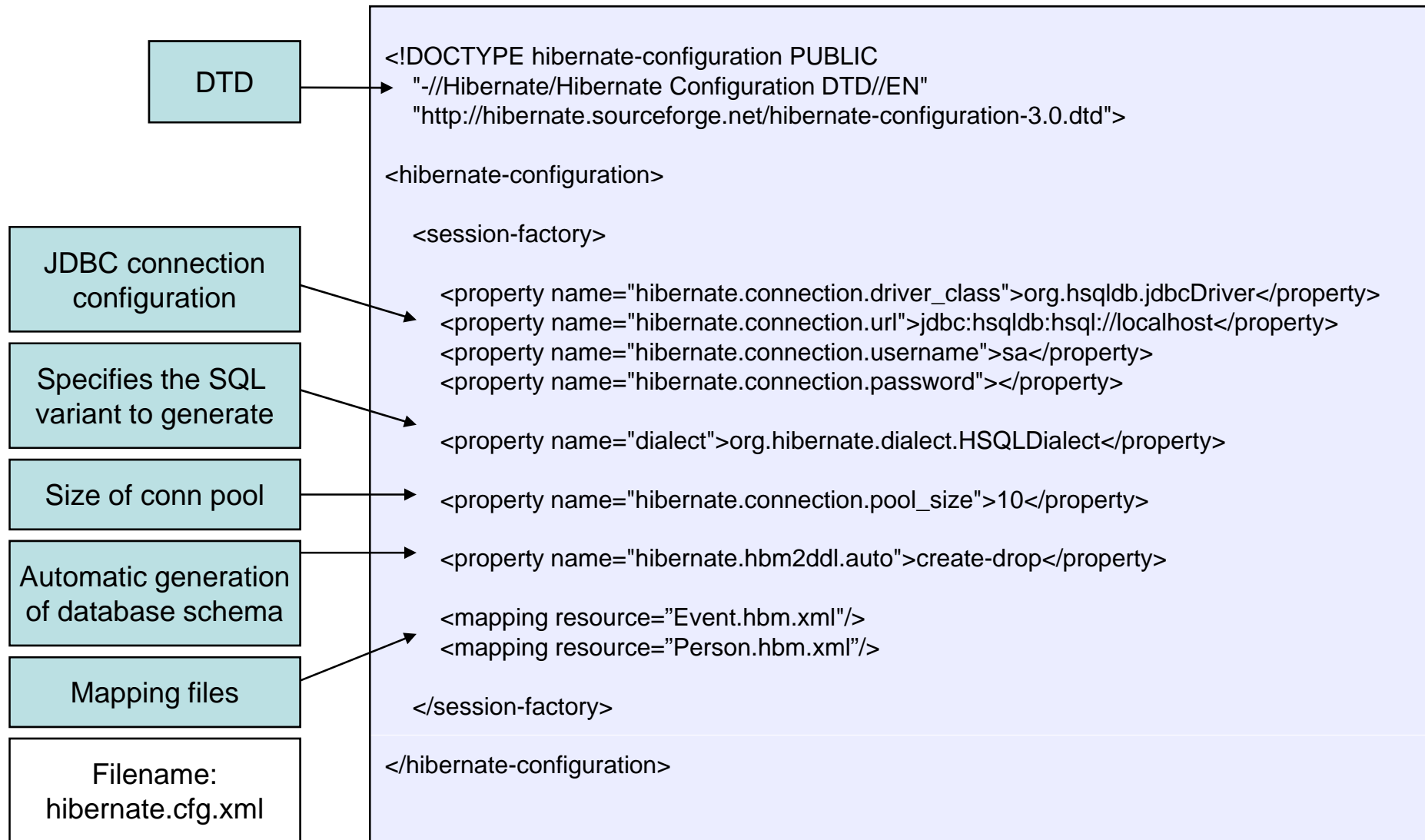
- Hibernate will translate Java types to SQL / database types for the properties of your mapped classes

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

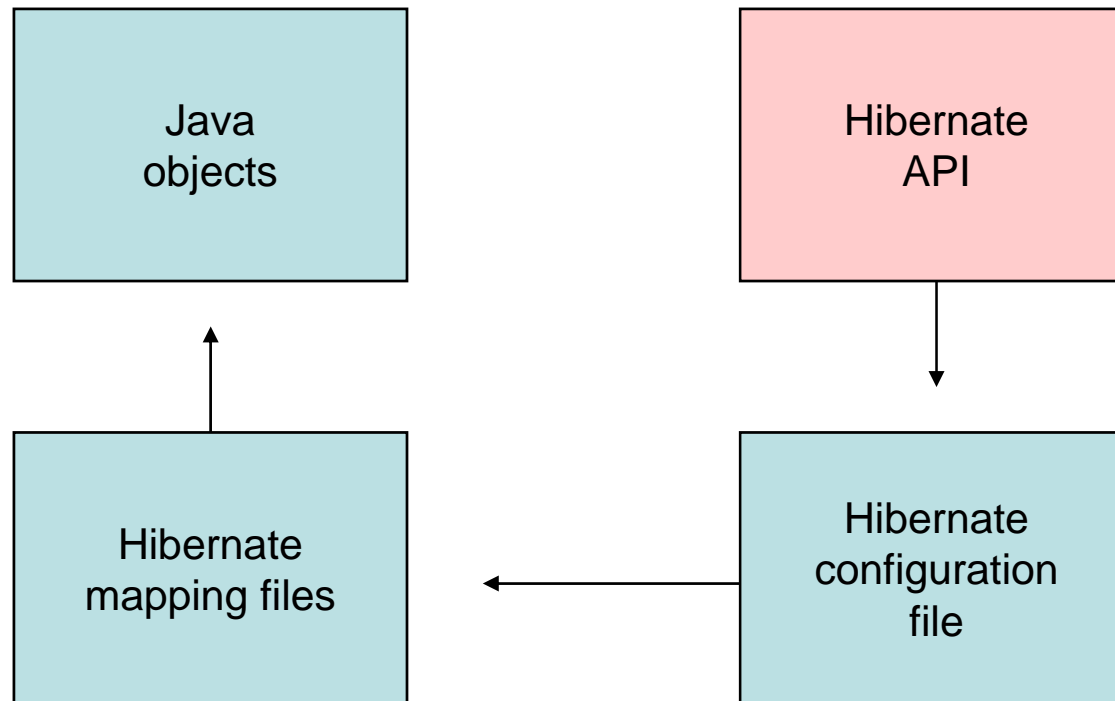
Example app: The EventManager



The Hibernate configuration file



Example app: The EventManager



The Configuration class

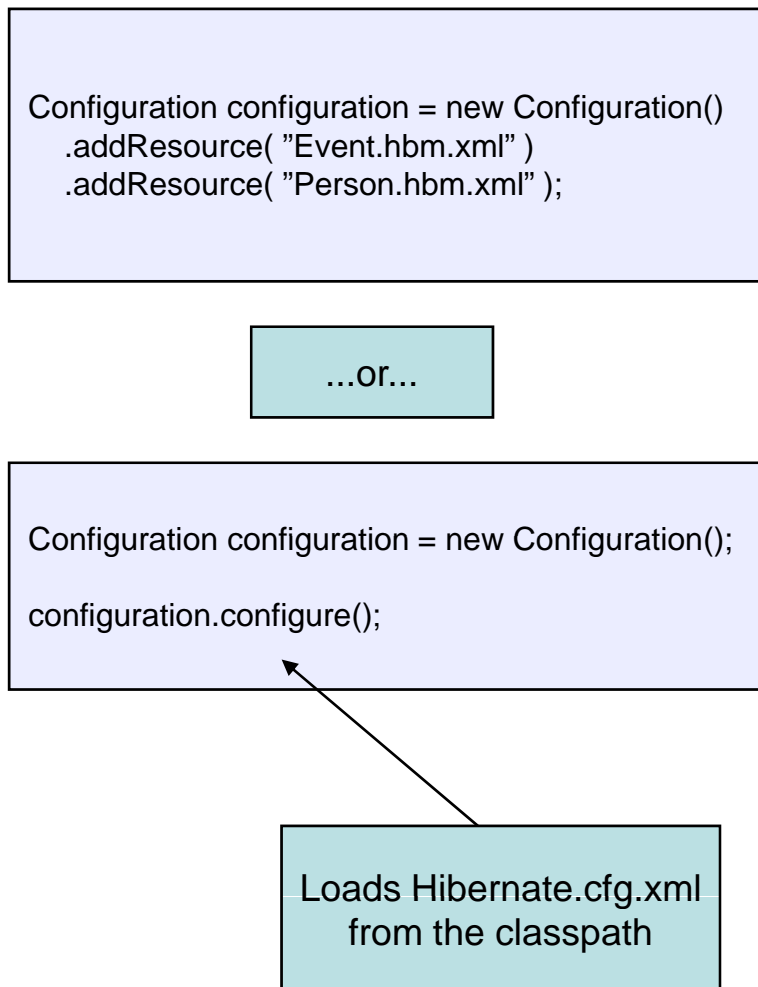
- Represents a set of mapping files
- Mapping files can be specified programmatically or through the Hibernate configuration file
- Intended as a startup-time object

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

...or...

```
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

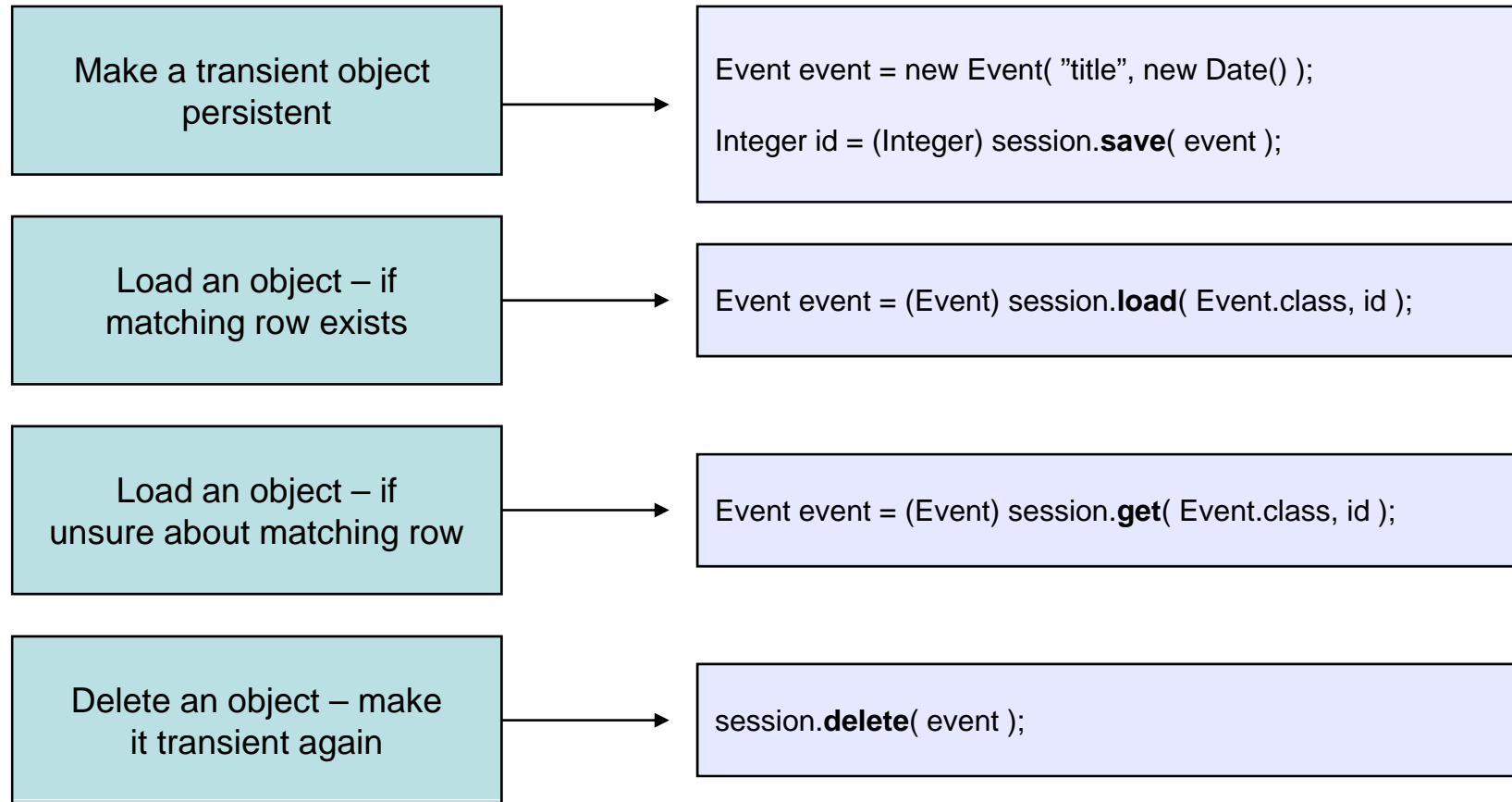
- Obtained from a SessionFactory instance
- Main runtime interface between a Java application and Hibernate
- Responsible for storing and retrieving objects
- Think of it as a collection of loaded objects related to a *single unit of work*

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

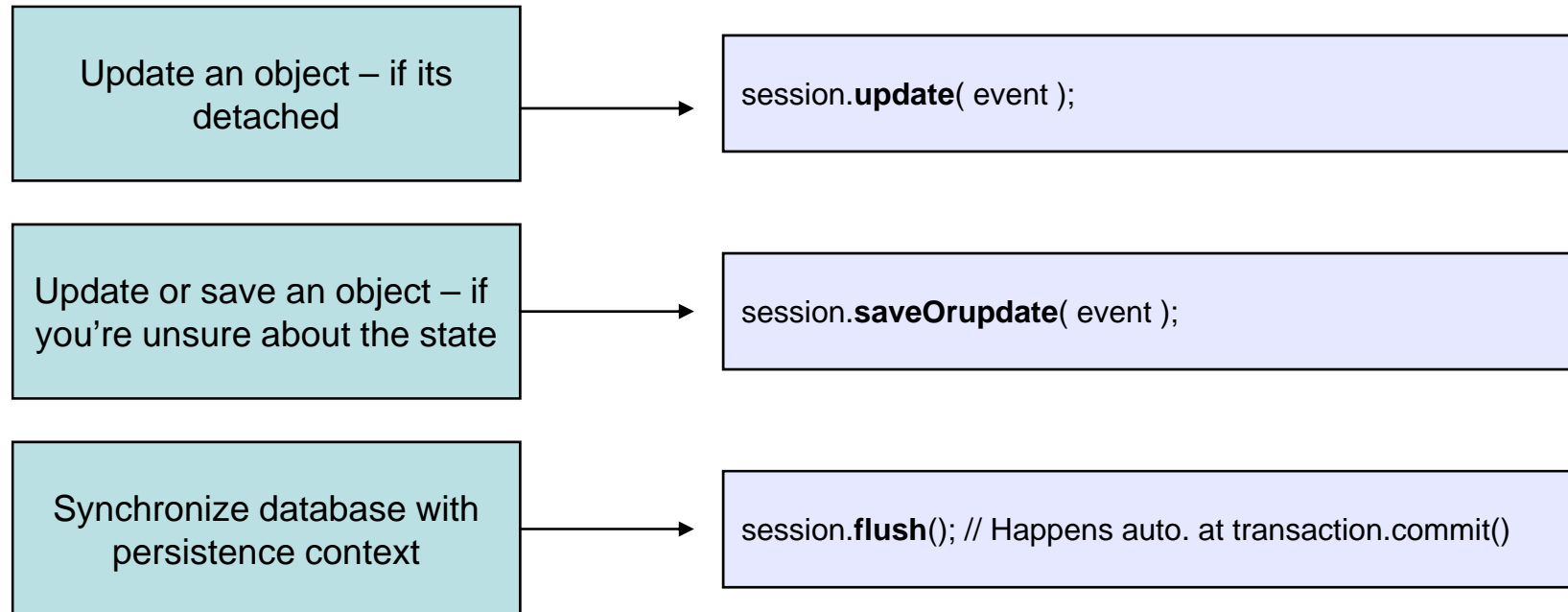
Instance states

- 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

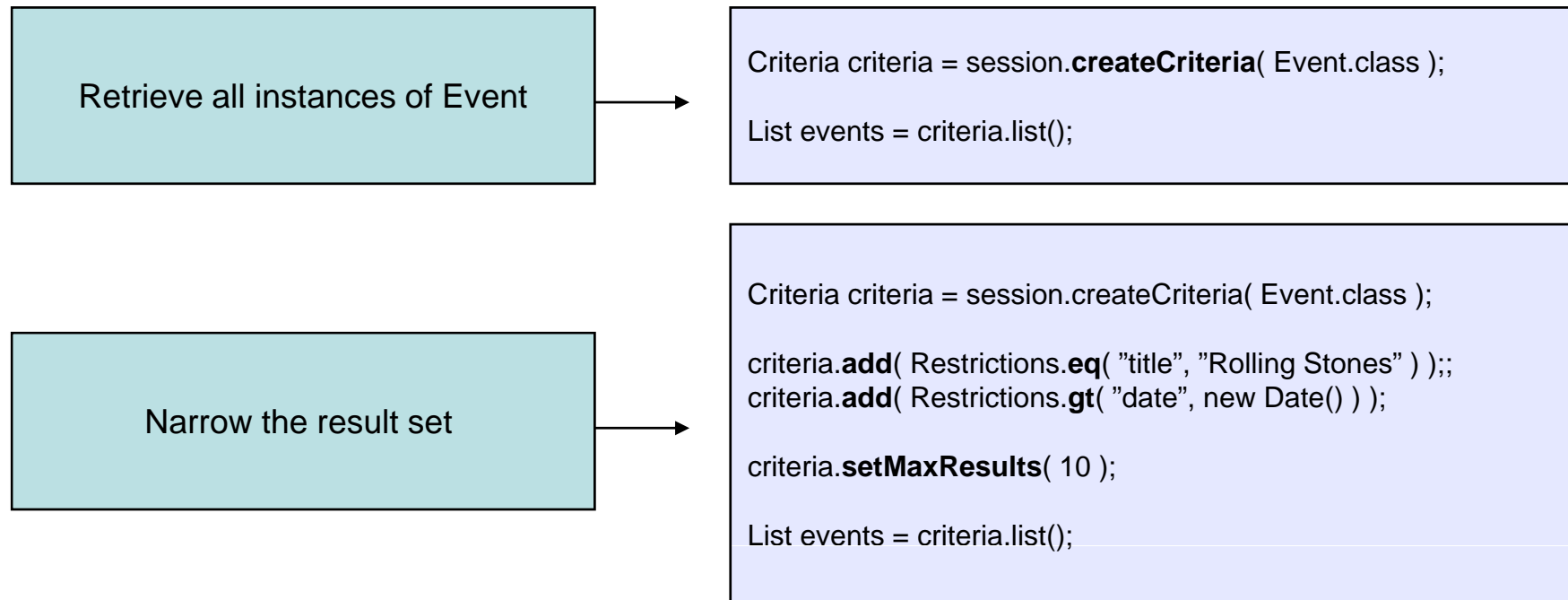


The Session interface



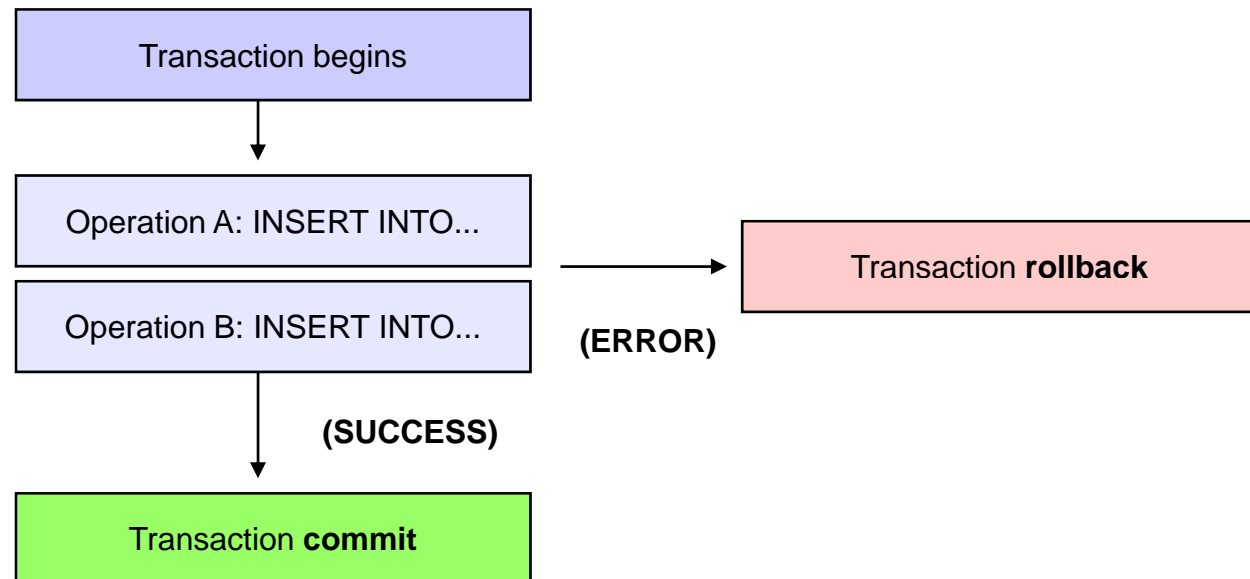
The Criteria interface

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



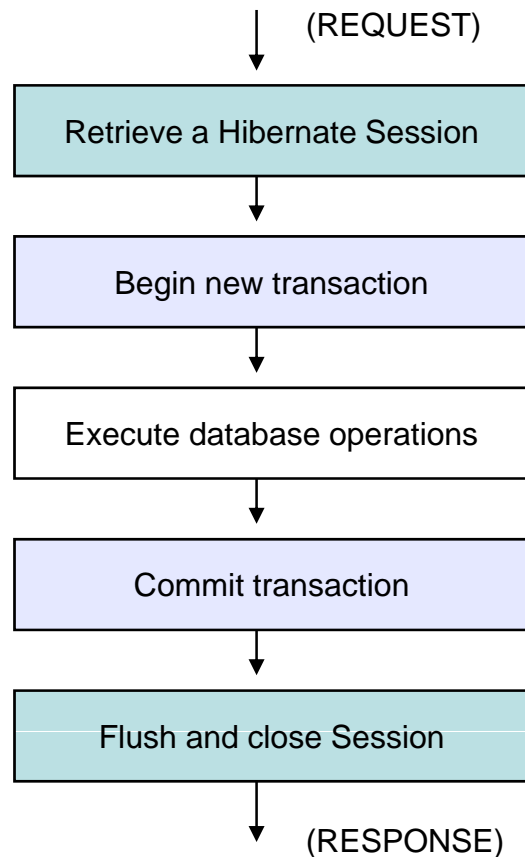
Transactions

- 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*



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

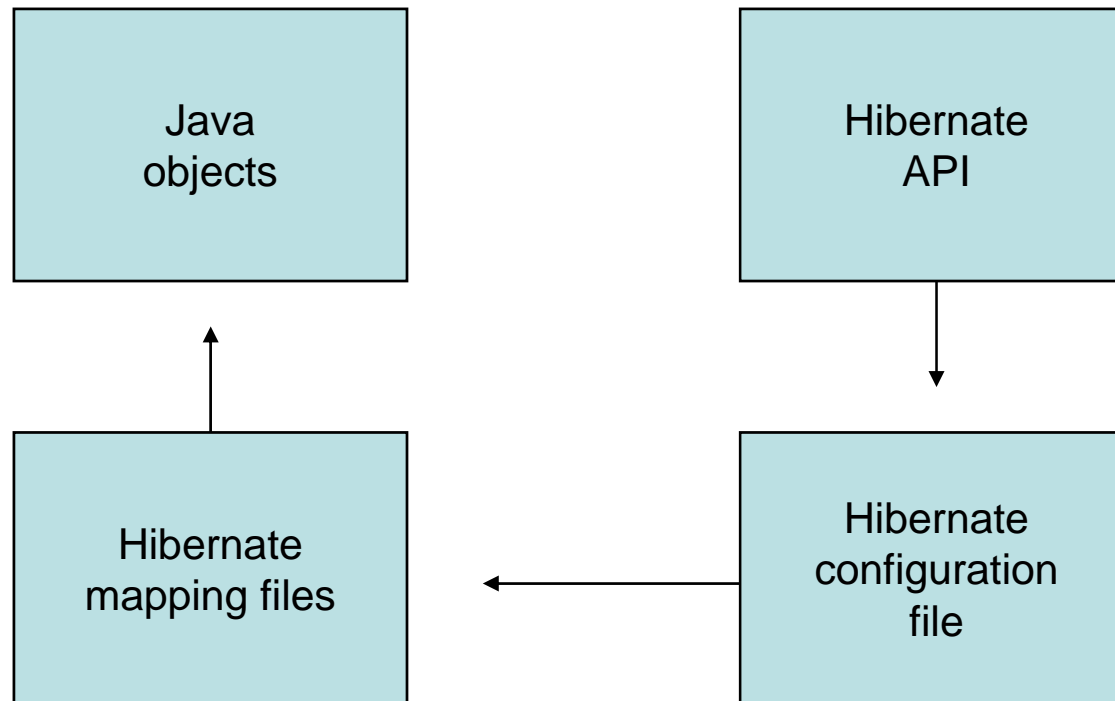
Transaction transaction = null;

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



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