Mappings and Queries

with

Hibernate

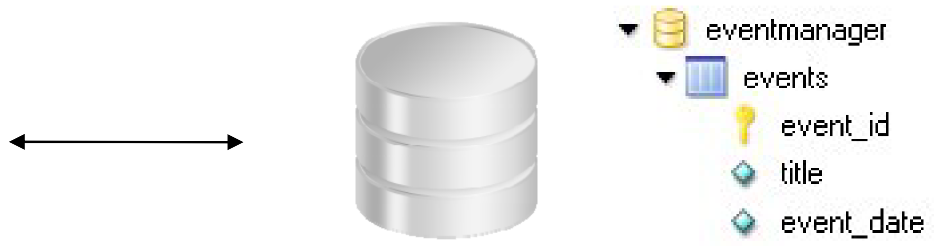
Mappings

- Collection mapping
- Association mapping
- Component mapping

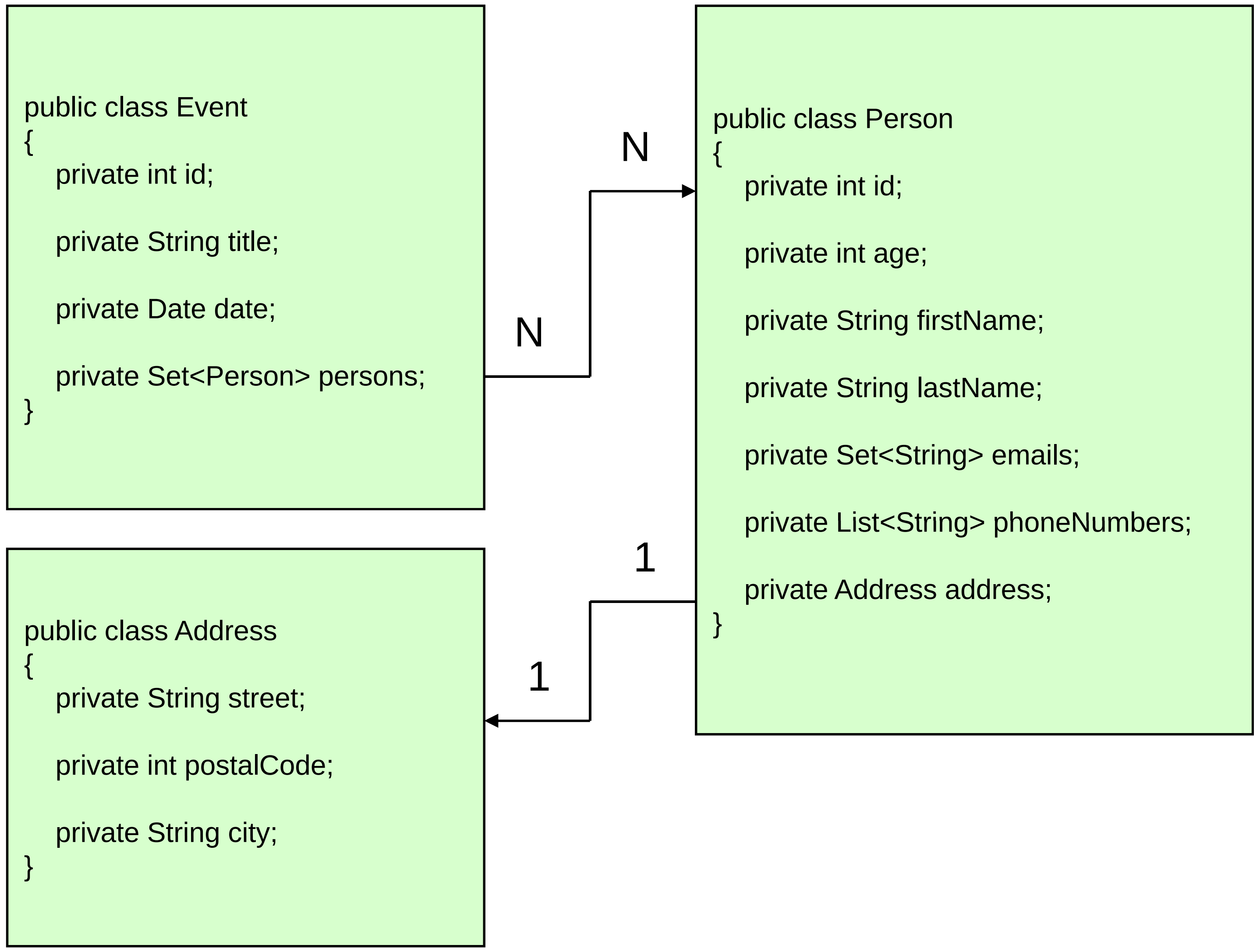
Revision

- Hibernate is an object-relational mapping framework
- Maps persistence operations between object models to relational databases
- Core elements in a Hibernate application are:
  - Your Java objects
  - The Hibernate object mapping files (Event.hbm.xml)
  - The Hibernate configuration file (Hibernate.cfg.xml)
  - Classes working with the Hibernate API (Session, Transaction)

```
public class Event
{
    private int id;
    private String title;
    private Date date;
    private Set<Person> persons;
}
```



Example: The EventManager



```
public class Event
{
    private int id;

    private String title;

    private Date date;

    private Set<Person> persons;
}
```

```
public class Person
{
    private int id;

    private int age;

    private String firstName;

    private String lastName;

    private Set<String> emails;

    private List<String> phoneNumbers;

    private Address address;
}
```

```
public class Address
{
    private String street;

    private int postalCode;

    private String city;
}
```

Collection mapping

- Collection properties must be declared as an interface type (Set, not HashSet)
- Hibernate provides built-in mapping for Set, Map, List, and more
- May contain basic types, custom types and references to other Hibernate objects
- Collections are represented by a *collection table* in the database
  - Collection key: foreign key of owning object
  - Collection element: object in the collection

Collection mapping

Java object

Collection declared as interface type

```
public class Person
{
    private int id;

    // other properties

    private Set<String> emails;

    // get and set methods
}
```

Hibernate mapping

Refers to Java property

Foreign key to owner

Actual content of set

```
<class name=”no.uio.inf5750.example.model.Person table=”persons”>

    <!-- other properties -->

    <set name=”emails” table=”person_emails”>
        <key column=”person_id”/>
        <element column=”email” type=”string”/>
    </set>

</class>
```

Indexed collections

- All *ordered* collection mappings need an *index column* in the collection table to persist the sequence
- Index of List is always of type Integer, index of Map can be of any type

Indexed collection mapping

List is an ordered type of collection

```
public class Person
{
    private int id;

    // other properties

    private List<String> phoneNumbers;

    // get and set methods
}
```

List mapped to table

Required mapping of index column

```
<class name="no.uio.inf5750.example.model.Person table="persons">

    <!-- other properties -->

    <list name="phoneNumbers" table="phone_numbers">
        <key column="person_id"/>
        <list-index column="sort_order" base="0"/>
        <element column="phone_number" type="string"/>
    </list>

</class>
```

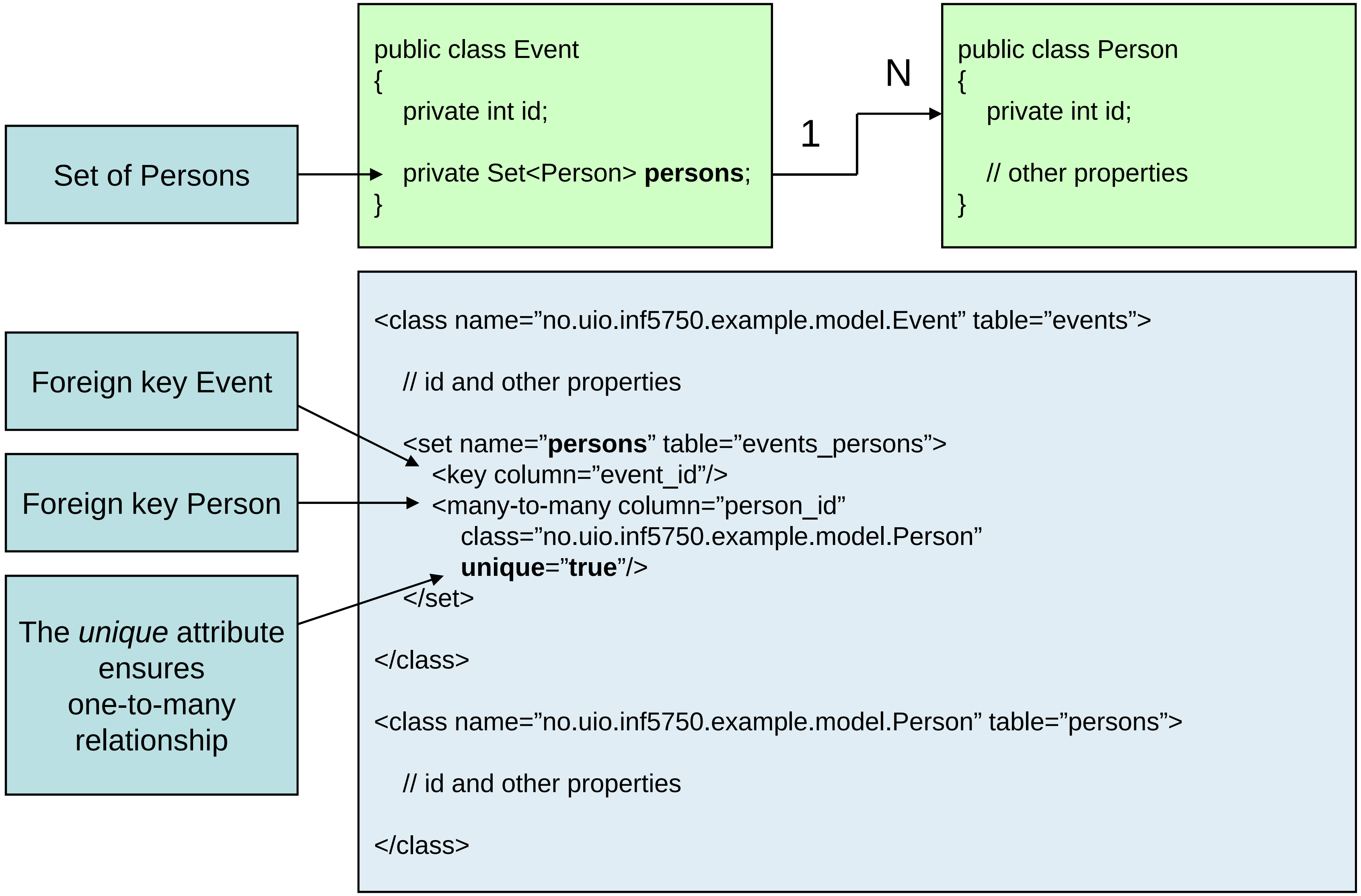
Association mapping

- Hibernate lets you easily specify all kinds of associations between objects
  - Unidirectional one-to-many
  - Unidirectional many-to-many
  - Bidirectional one-to-many
  - Bidirectional many-to-many

- Representing associations with join tables makes the database schema cleaner

- Nullable foreign keys bad practise

Unidirectional one-to-many

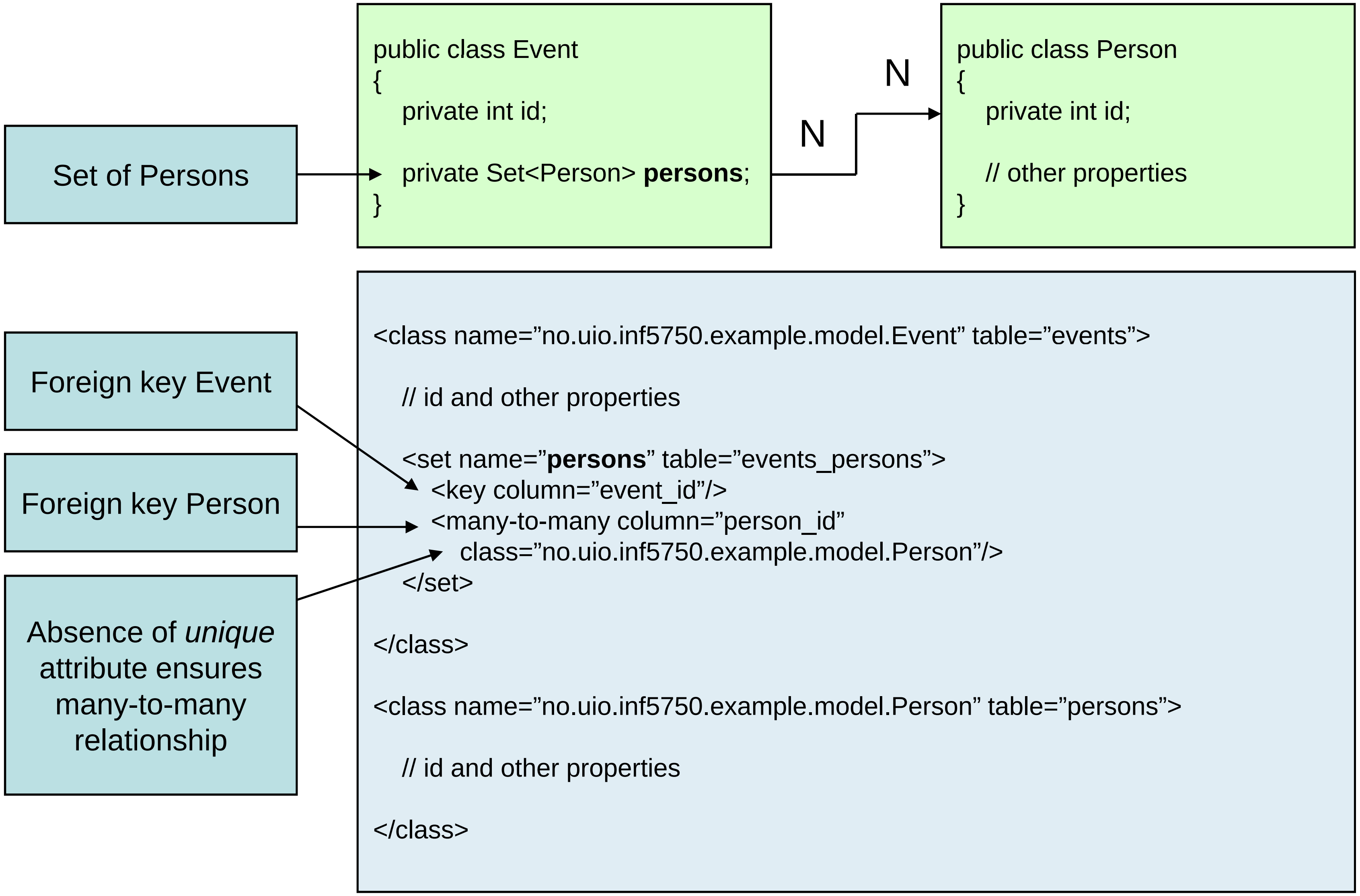


```
public class Event
{
    private int id;

    private Set<Person> persons;
}
```

```
public class Person
{
    private int id;

    // other properties
}
```

```
<class name=”no.uio.inf5750.example.model.Event” table=”events”>

    // id and other properties

    <set name=”persons” table=”events_persons”>
        <key column=”event_id”/>
        <many-to-many column=”person_id”
            class=”no.uio.inf5750.example.model.Person”
            unique=”true”/>
    </set>

</class>

<class name=”no.uio.inf5750.example.model.Person” table=”persons”>

    // id and other properties

</class>
```

Unidirectional many-to-many


Set of Persons
N
N
Foreign key Event
Foreign key Person
Absence of *unique* attribute ensures many-to-many relationship

```
public class Event
{
    private int id;

    private Set<Person> persons;
}
```

```
public class Person
{
    private int id;

    // other properties
}
```

```
<class name=”no.uio.inf5750.example.model.Event” table=”events”>

    // id and other properties

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

</class>

<class name=”no.uio.inf5750.example.model.Person” table=”persons”>

    // id and other properties

</class>
```

Bidirectional one-to-many

Event reference...

Set of Persons

```
public class Event
{
    private int id;

    private Set<Person> persons;
}
```

1

N

```
public class Person
{
    private int id;

    private Event event;
}
```

The *unique* attribute ensures one-to-many relationship

Specifies which join table to use for the association

Refers to property in Java class

```
<class name=”no.uio.inf5750.example.model.Event” table=”events”>

    <set name=”persons” table=”events_persons”>
        <key column=”event_id”/>
        <many-to-many column=”person_id”
            class=”no.uio.inf5750.example.model.Person” unique=”true”/>
    </set>

</class>

<class name=”no.uio.inf5750.example.model.Person” table=”persons”>

    <join table=”events_persons” inverse=”true”>
        <key column=”person_id”/>
        <many-to-one column=”event_id” name=”event”/>
    </join>

</class>
```

Bidirectional many-to-many

Set of Events...

Set of Persons

```
public class Event
{
    private int id;

    private Set<Person> persons;
}
```

N

N

```
public class Person
{
    private int id;

    private Set<Event> events;
}
```

Absence of unique attribute ensures many-to-many relationship

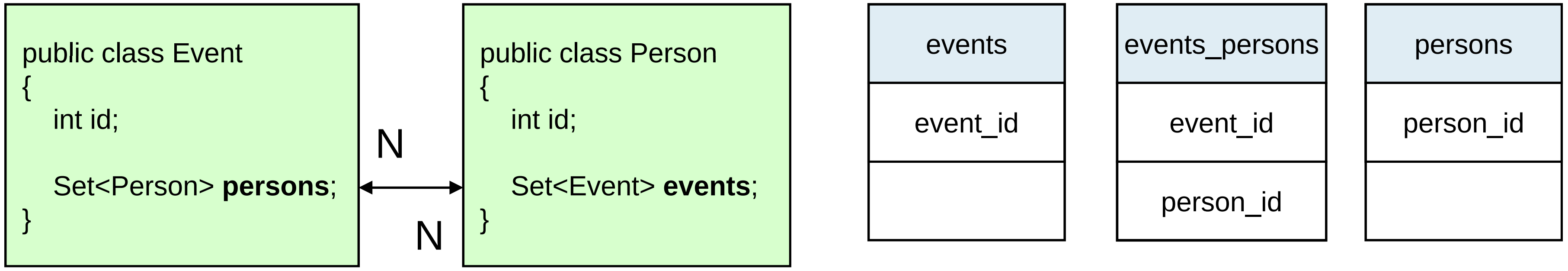
Key and many-to-many value swapped

Both sides can be inverse in many-to-many associations

```
<class name=”no.uio.inf5750.example.model.Event” table=”events”>

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

</class>

<class name=”no.uio.inf5750.example.model.Person” table=”persons”>

    <set name=”events” table=”events_persons” inverse=”true”>
        <key column=”person_id”/>
        <many-to-many column=”event_id”
            class=”no.uio.inf5750.example.model.Event”/>
    </set>

</class>
```

The *inverse* property explained

- Bidirectional associations must be updated *on both sides* in the Java code!
- Hibernate maps many-relationships with a *join table*
- Hibernate must ignore one side to avoid constraint violations!
- Must be *many*-side on one-to-many, doesn’t matter on many-to-many


```
public class Event
{
    int id;

    Set<Person> persons;
}
```

```
public class Person
{
    int id;

    Set<Event> events;
}
```

events	events_persons	persons
event_id	event_id	person_id
	person_id	

Component mapping

- A component is an object saved as a value, not as a reference
- Saved directly – no need to declare interfaces or identifiers
- Required to define an empty constructor
- Shared references not supported

Component mapping

Component

```
public class Address
{
    private String street;
    private int postalCode;
    private String city;

    // no-arg constructor, get/set
}
```

```
public class Person
{
    // other properties

    private Address address;

    // get and set methods
}
```

Component mapping

Property mapping

```
<class name="no.uio.inf5750.example.model.Person table="persons">

    <!-- other properties -->

    <component name="address">
        <property name="street"/>
        <property name="postalCode"/>
        <property name="city"/>
    </component>

</class>
```

Queries

- The Query interface
- The Hibernate Query Language (HQL)

The Query interface

- You need a *query* when you don’t know the identifiers of the objects you are looking for
- Used mainly to execute Hibernate Query Language queries
- Obtained from a Hibernate Session instance
- Provides functionality for:
  - Parameter binding to *named query parameters*
  - Retrieving lists of objects or unique objects
  - Limiting the number of retrieved objects

```
Query query = session.createQuery( ”some_HQL_query” );
```

The Hibernate Query Language

- HQL is an *object-oriented* query language
  - Syntax has similarities to SQL
  - Not working agains tables and columns, but objects!
- Understands object-oriented concepts like inheritance
- Has advanced features like:
  - Associations and joins
  - Polymorphic queries
  - Subqueries
  - Expressions
- Reduces the size of queries

The from clause

Simplest possible query, qualified class name auto-imported, will return all Person instances:

```
from Person
```

Convenient to assign an alias to refer to in other parts of the query:

```
from Person as p
```

Multiple classes may be desired. The alias keyword is optional:

```
from Person p, Event e
```

The where clause

Allows you to narrow the returned list, properties can be referred to by name:

```
from Person where firstName='John'
```

If there is an alias, use a qualified property name:

```
from Person p where p.lastName='Doe'
```

Compound path expressions is powerful:

```
from Person p where p.address.city='Boston'
```

Expressions

In clause:

```
from Person p where p.firstName in ( ’John’, ’Tom’, ’Greg’ )
```

Between and *not* clause:

```
from Person p where p.lastName not between ’D’ and ’F’
```

Size clause:

```
from Person p where size ( p.address ) > 2
```

Query examples

HQL query with *named query parameter* (age)

Query obtained from Session

Age parameter binding

Max nr of objects restriction

Returns the result as a List

```
public Collection<Person> getPersonsByAge( int age, int maxResults )
{
    Session session = // retrieve a session instance somehow

    String hql = "from Person where age = :age";

    Query query = session.createQuery( hql );

    query.setInteger( "age", age );

    query.setMaxResults( maxResults );

    return query.list();
}
```

Query examples

HQL query with *named query parameters*

Create query and pass in HQL string as parameter

Parameter binding with the setString methods

uniqueResult offers a shortcut if you know a single object will be returned

```
public Person getPerson( String firstName, String lastName )
{
    Session session = // retrieve a session instance somehow

    String hql = "from Person where firstName = :firstName ” +
        ”and lastName = :lastName";

    Query query = session.createQuery( hql );

    query.setString( "firstName", firstName );

    query.setString( "lastName", lastName );

    return (Person) query.uniqueResult();
}
```

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