Lightweight J2EE Framework

Struts, spring, hibernate

Software System Design
Zhu Hongjun

Session 5: Hibernate DAO

- Transaction Management and Concurrency
- Hibernate Querying
- Batch Processing
- Data Filtering
- Interceptors and Events

Transaction Management and Concurrency

- Hibernate directly uses JDBC connections and JTA resources without adding any additional locking behavior
- Hibernate does not lock objects in memory. Your application can expect the behavior as defined by the isolation level of your database transactions

Transaction Management and Concurrency

- Session and Transaction Scopes
  - A SessionFactory is an expensive-to-create, threadsafe object, intended to be shared by all application threads. It is created once, usually on application startup, from a Configuration instance
  - A Session is an inexpensive, non-threadsafe object that should be used once and then discarded for: a single request, a conversation or a single unit of work

Transaction Management and Concurrency

- Session and Transaction Scopes (cont.)
  - In order to reduce lock contention in the database, a database transaction has to be as short as possible
  - Unit of work
    - A series of operations we wish to carry out against the database together
    - Do not use the *session-per-operation* antipattern
    - The most common pattern in a multi-user client/server application is *session-per-request*

Transaction Management and Concurrency

- Session and Transaction Scopes (cont.)
  - ACID
    - Atomicity
      - One atomic unit of work, if one step fails, it all fails
    - Consistency
      - Works on a consistent set of data that is hidden from other concurrently running transaction
    - Isolation
      - A particular transaction should not be visible to other concurrently running transactions
    - Durability

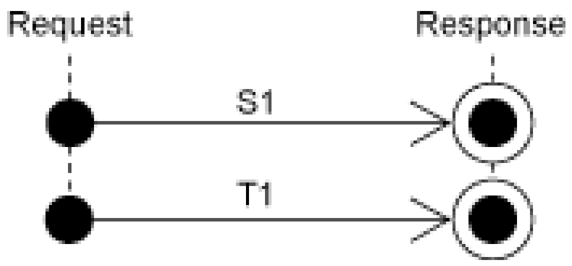
Transaction Management and Concurrency

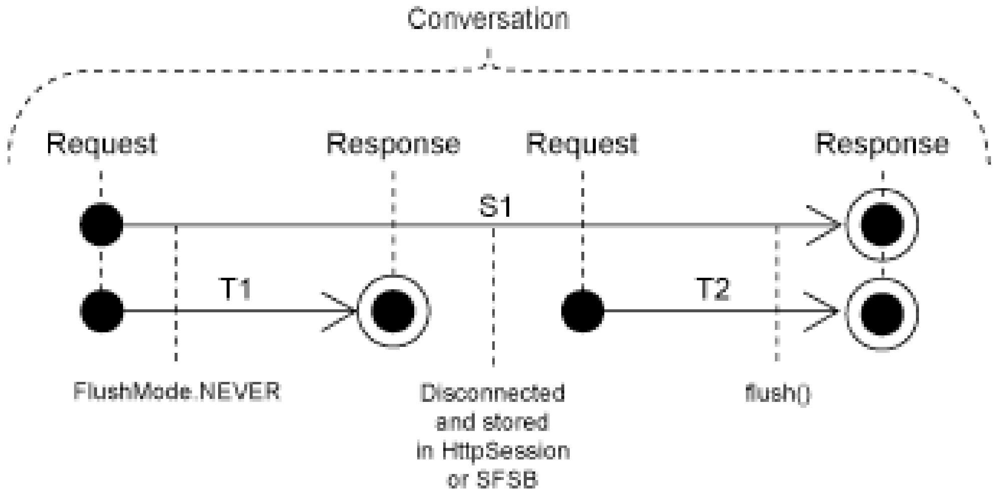
- Session and Transaction Scopes (cont.)
  - Hibernate provides built-in management of the "current session" to simplify *session-per-request*
    - Your application code can access a “current session” to process the request by calling sessionFactory.getCurrentSession()
  - You can extend the scope of a Session and database transaction until the "view has been rendered“——Open Session In View

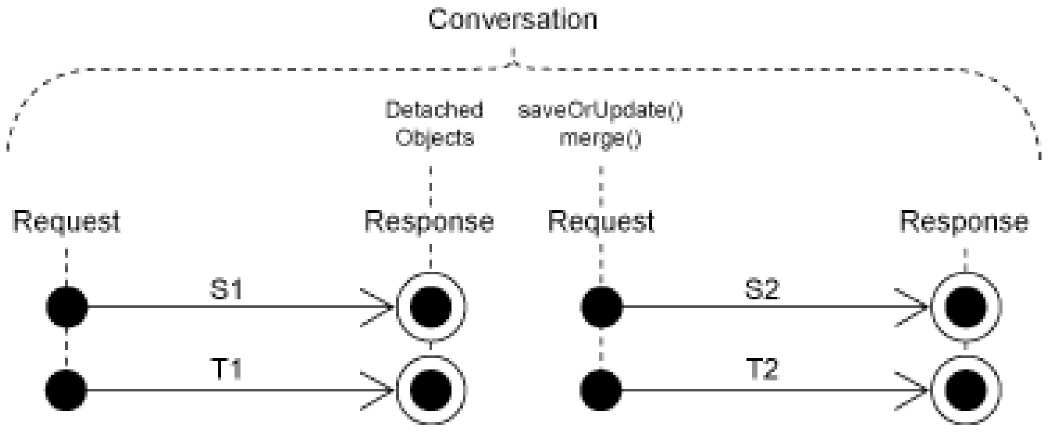
Transaction Management and Concurrency

- Session and Transaction Scopes (cont.)
  - Long conversations
    - Many business processes require a whole series of interactions with the user that are interleaved with database accesses
    - Ways to implement by hibernate features
      - *Automatic Versioning*
      - *Detached Objects*
      - *Extended (or Long) Session*

Scope of unit-of-work

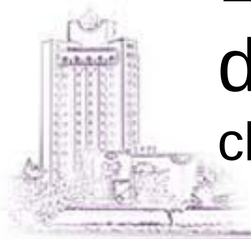






Transaction Management and Concurrency

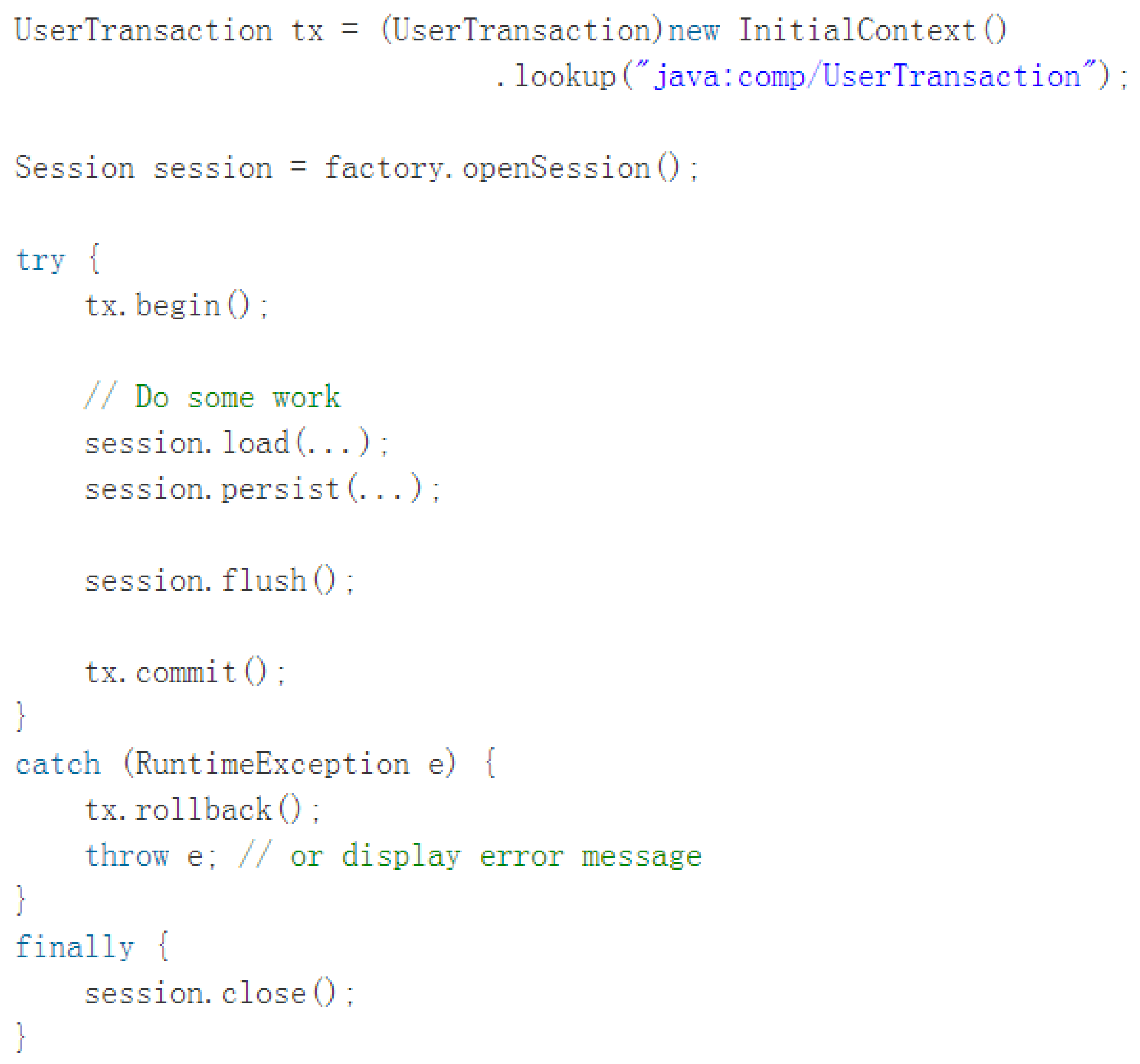
- Database transaction demarcation
  - Database, or system, transaction boundaries are always necessary. No communication with the database can occur outside of a database transaction
  - A Hibernate application can run in non-managed and managed J2EE environments.
  - Ending a Session usually involves four distinct phases: flush the session, commit the transaction, close the session, handle exceptions



Transaction Management and Concurrency

- Database transaction demarcation (cont.)
  - Exception Handling
    - If the Session throws an exception, immediately rollback the database transaction, call Session.close() and discard the Session instance
    - The standard JDBCException subtypes are
      - JDBCConnectionException
      - SQLGrammarException
      - ConstraintViolationException
      - LockAcquisitionException
      - GenericJDBCException

Transaction demarcation with JTA



```
UserTransaction tx = (UserTransaction)new InitialContext()
                            .lookup("java:comp/UserTransaction");

Session session = factory.openSession();

try {
    tx.begin();

    // Do some work
    session.load(...);
    session.persist(...);

    session.flush();

    tx.commit();
}
catch (RuntimeException e) {
    tx.rollback();
    throw e; // or display error message
}
finally {
    session.close();
}
```

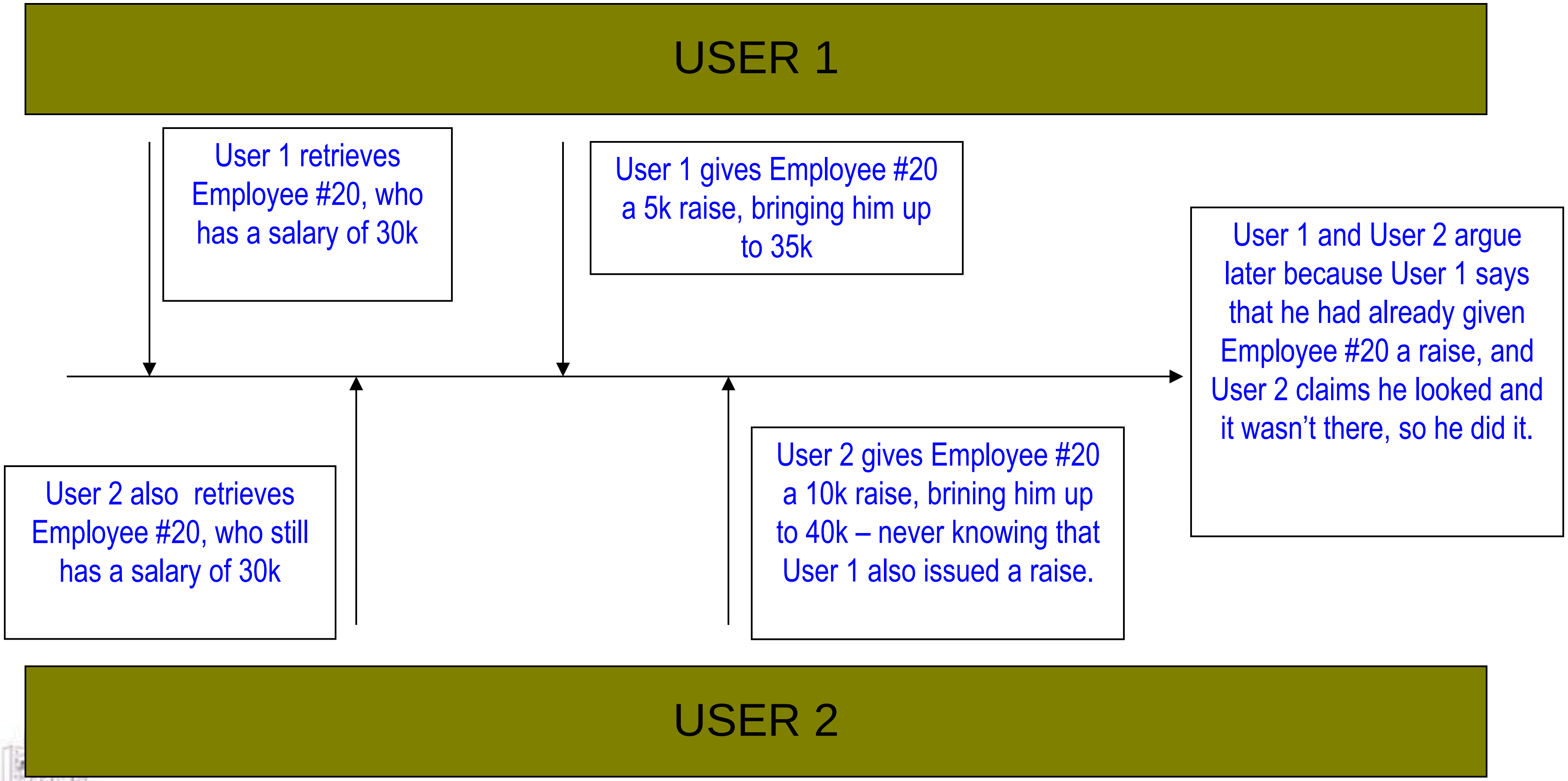
Transaction demarcation with JDBC

```
try {
    factory.getCurrentSession().beginTransaction();

    // Do some work
    factory.getCurrentSession().load(...);
    factory.getCurrentSession().persist(...);

    factory.getCurrentSession().getTransaction().commit();
}
catch (RuntimeException e) {
    factory.getCurrentSession().getTransaction().rollback();
    throw e; // or display error message
}
```

Transaction Management and Concurrency

- Concurrency
  - System allows multiple users to work with and modify the same database record simultaneously
  - Who ever updates last, wins!
  - Almost always overlooked by developers
    - Difficult to reproduce or validate a customer complaint
    - No record of anything failing
    - Needs to be pro-actively thought of

Overwriting Other Users Scenario Demo


USER 1

User 1 retrieves Employee #20, who has a salary of 30k

User 1 gives Employee #20 a 5k raise, bringing him up to 35k

User 1 and User 2 argue later because User 1 says that he had already given Employee #20 a raise, and User 2 claims he looked and it wasn't there, so he did it.

User 2 also retrieves Employee #20, who still has a salary of 30k

User 2 gives Employee #20 a 10k raise, brining him up to 40k – never knowing that User 1 also issued a raise.

USER 2

Transaction Management and Concurrency

- Concurrency (cont.)
  - Preventing Overwriting By Locking Records
    - Pessimistic
      - Don’t allow other users to even read the record while you have it “checked out”
      - We’re pessimistic, believing someone will change it on us
    - Optimistic
      - Everyone can access it, and we check to see if it’s been updated when we commit changes
      - If it’s been changed while a user was working on it, alert them so we can act accordingly

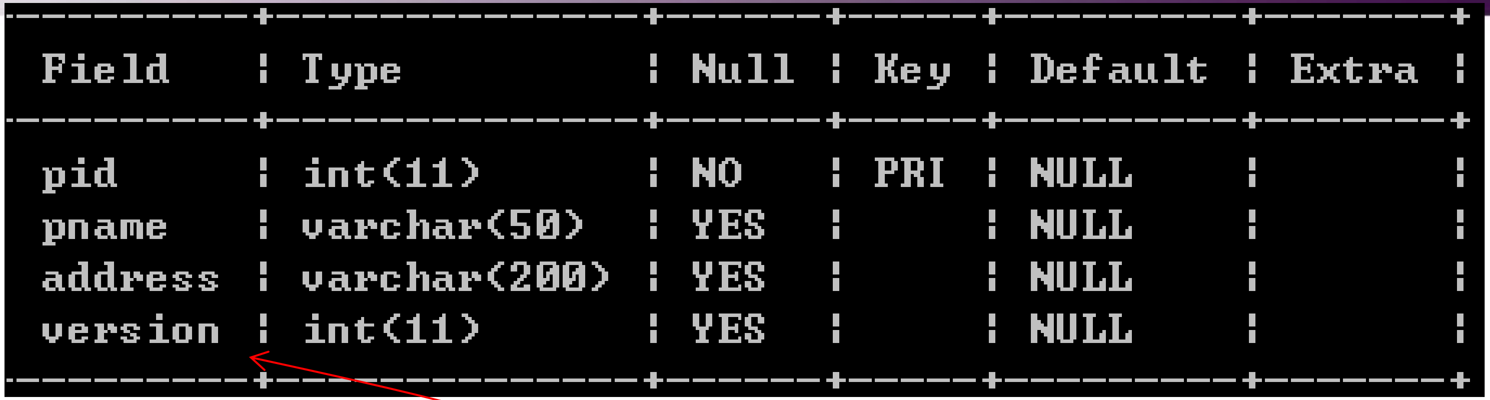
Transaction Management and Concurrency

- Optimistic concurrency control
  - The only approach that is consistent with high concurrency and high scalability, is optimistic concurrency control with versioning
  - Version checking uses version numbers, or timestamps, to detect conflicting updates and to prevent lost updates
    - Automatic versioning

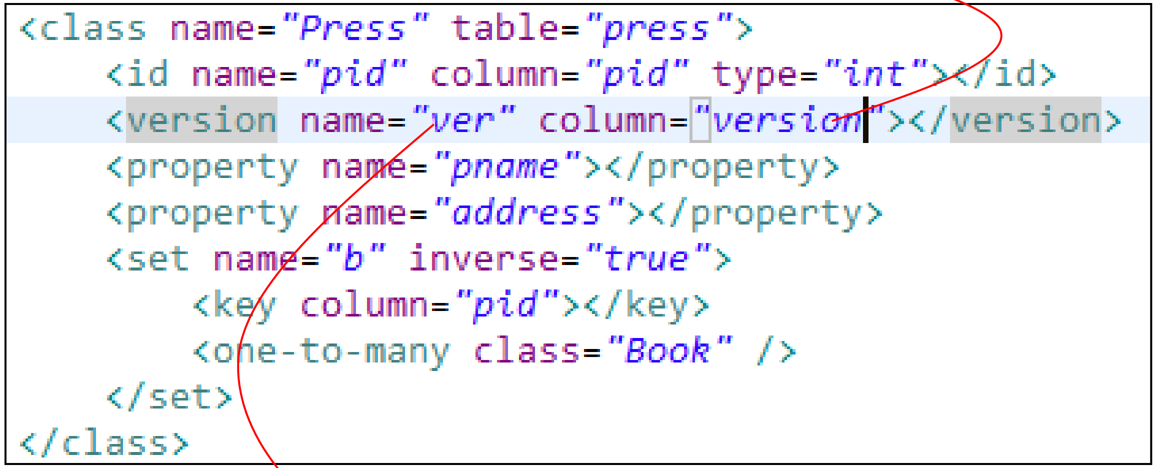
Transaction Management and Concurrency

- Optimistic concurrency control (cont.)
  - Automatic versioning
    - Decide on a versioning strategy
    - Add an attribute to every domain object Java class to represent the version
    - Add a column to every domain object database table to represent the version
    - Update the object mapping files to make Hibernate aware of the version attribute by <version/> element

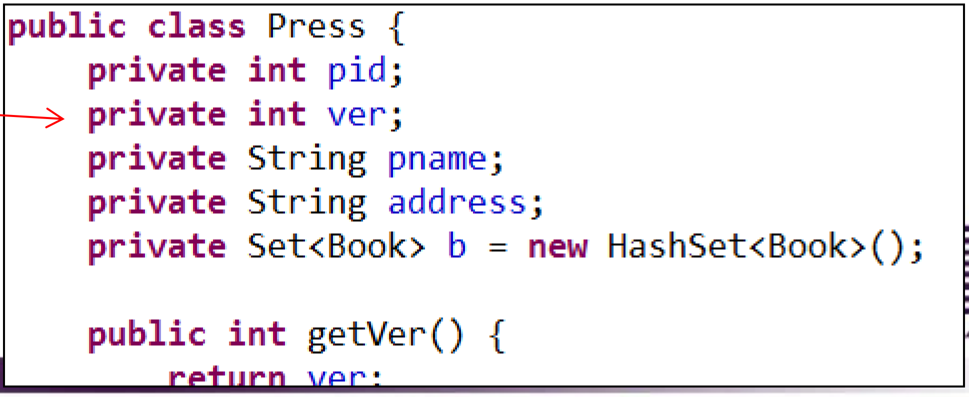
Auto Versioning Demo



```
+----------+----------------+------+-----+---------+-------+
| Field    | Type           | Null | Key | Default | Extra |
+----------+----------------+------+-----+---------+-------+
| pid      | int(11)        | NO   | PRI | NULL    |       |
| pname    | varchar(50)    | YES  |     | NULL    |       |
| address  | varchar(200)   | YES  |     | NULL    |       |
| version  | int(11)        | YES  |     | NULL    |       |
+----------+----------------+------+-----+---------+-------+
```



```
<class name="Press" table="press">
    <id name="pid" column="pid" type="int"></id>
    <version name="ver" column="version"></version>
    <property name="pname"></property>
    <property name="address"></property>
    <set name="b" inverse="true">
        <key column="pid"></key>
        <one-to-many class="Book" />
    </set>
</class>
```



```
public class Press {
    private int pid;
    private int ver;
    private String pname;
    private String address;
    private Set<Book> b = new HashSet<Book>();

    public int getVer() {
        return ver;
```

Transaction Management and Concurrency

- Optimistic concurrency control (cont.)
  - Customizing auto versioning
    - Setting the optimistic-lock mapping attribute
      - False
        - Hibernate will then no longer increment versions if the property/collection is dirty
      - All
        - To force a version check with a comparison of the state of all fields in a row but without a version or timestamp property mapping in the <class>
      - Dirty
        - Hibernate will only compare dirty fields during flush, when mapping the <class>

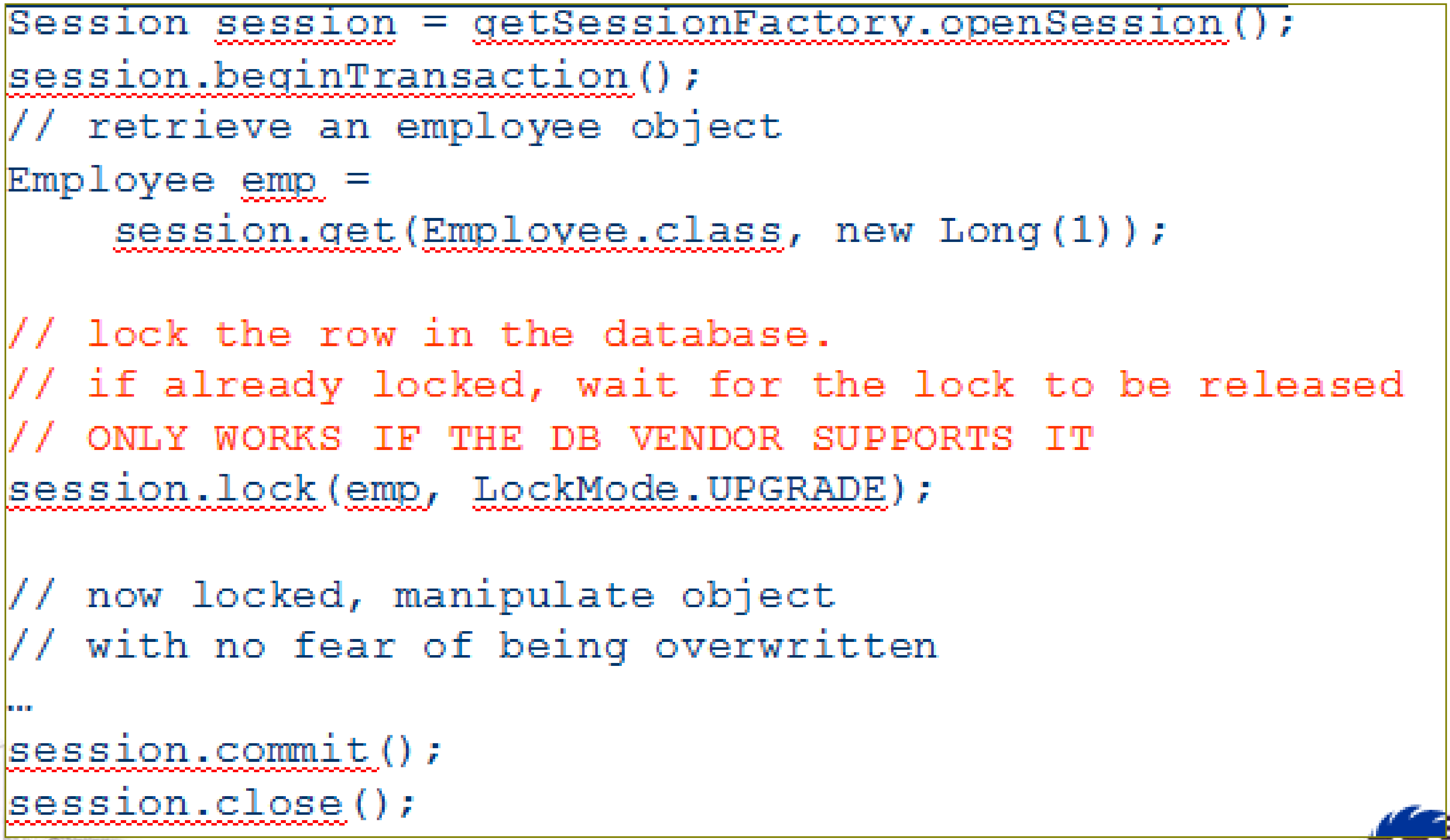
Transaction Management and Concurrency

- Pessimistic locking
  - Hibernate will always use the locking mechanism of the database; it never lock objects in memory
  - The LockMode class defines the different lock levels that can be acquired by Hibernate. A lock is obtained by the following mechanisms
    - LockMode.WRITE, LockMode.UPGRADE, LockMode.READ
    - LockMode.NONE

Transaction Management and Concurrency

- Pessimistic locking (cont.)
  - Using pessimistic lock
    - No version or timestamp attributes/columns required
    - Will not work for conversations
    - Does not scale as well
    - The "explicit user request" is expressed in one of the following ways
      - Session.load()/get()，specifying a LockMode
      - Session.lock()
      - Query.setLockMode()
      - **buildLockRequest**(lockOptions)

Pessimistic Locking Demo



```
Session session = getSessionFactory.openSession();
session.beginTransaction();
// retrieve an employee object
Employee emp =
    session.get(Employee.class, new Long(1));

// lock the row in the database.
// if already locked, wait for the lock to be released
// ONLY WORKS IF THE DB VENDOR SUPPORTS IT
session.lock(emp, LockMode.UPGRADE);

// now locked, manipulate object
// with no fear of being overwritten
…
session.commit();
session.close();
```

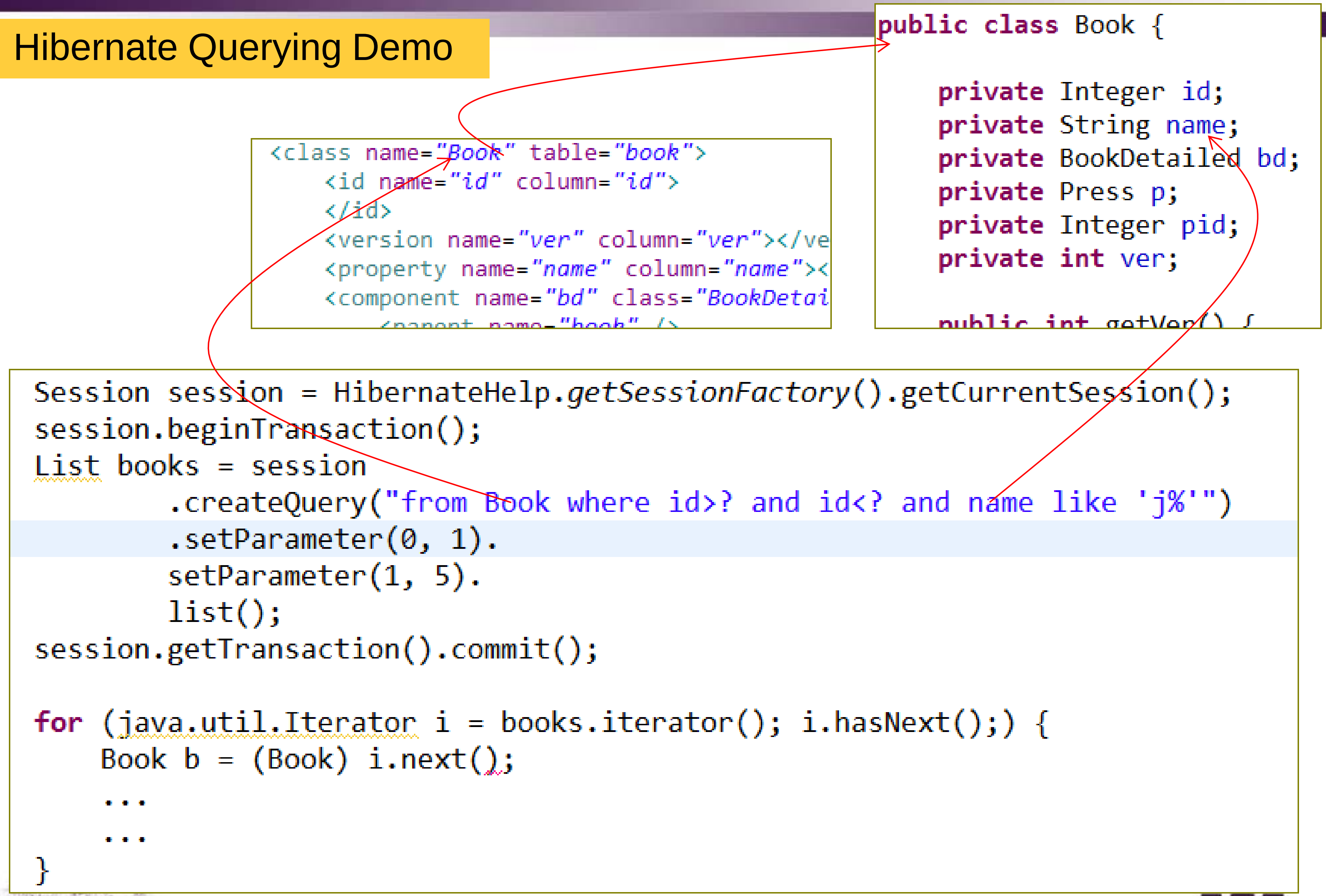
Hibernate Querying

- Hibernate supports an easy-to-use but powerful object oriented query language (HQL)
- For programmatic query creation, Hibernate supports a sophisticated Criteria and Example query feature (QBC and QBE)
- You can also express your query in the native SQL of your database

Hibernate Querying

- Executing queries
  - HQL and native SQL queries are represented with an instance of org.hibernate.Query
  - This interface offers methods for parameter binding, result set handling, and for the execution of the actual query.
  - You always obtain a Query using the current Session

Hibernate Querying Demo



```
public class Book {

    private Integer id;
    private String name;
    private BookDetailed bd;
    private Press p;
    private Integer pid;
    private int ver;

    public int getVer() {
```

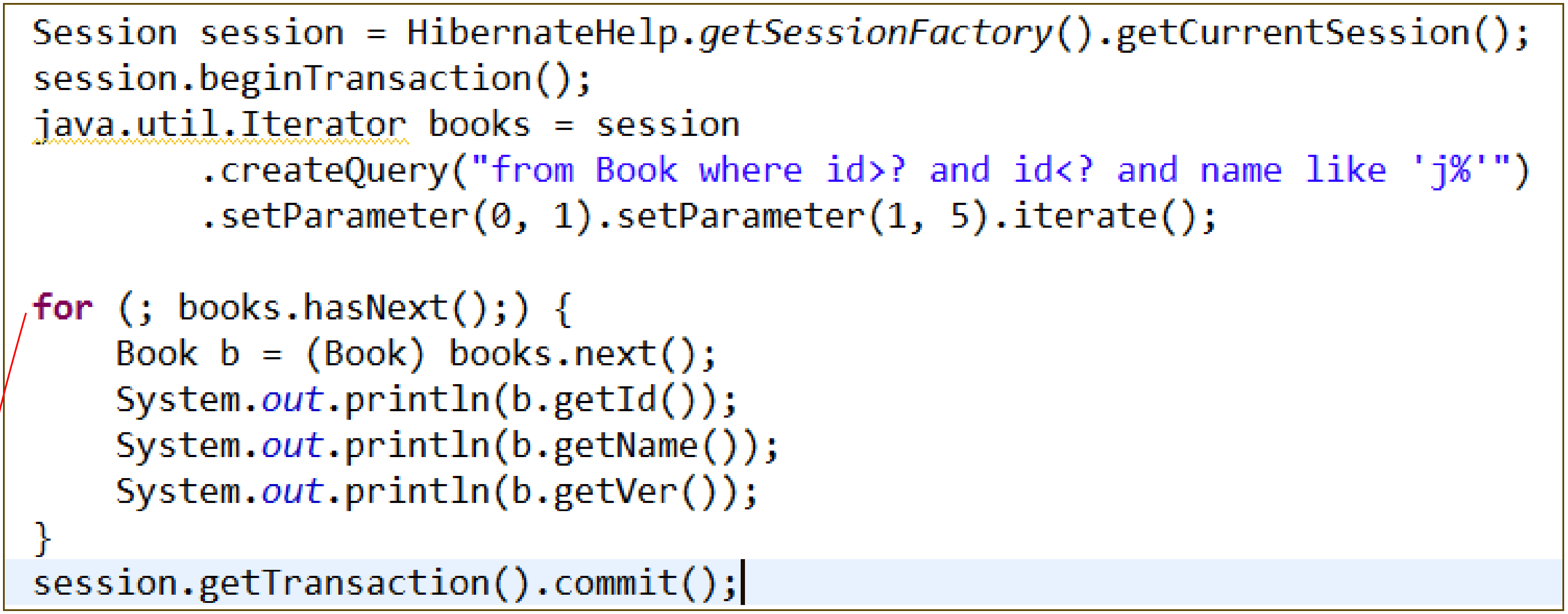
```
<class name="Book" table="book">
    <id name="id" column="id">
    </id>
    <version name="ver" column="ver"></ve
    <property name="name" column="name"><
    <component name="bd" class="BookDetai
        <parent name="book" />
```

```
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();
List books = session
        .createQuery("from Book where id>? and id<? and name like 'j%'")
        .setParameter(0, 1).
        setParameter(1, 5).
        list();
session.getTransaction().commit();

for (java.util.Iterator i = books.iterator(); i.hasNext();) {
    Book b = (Book) i.next();
    ...
    ...
}
```

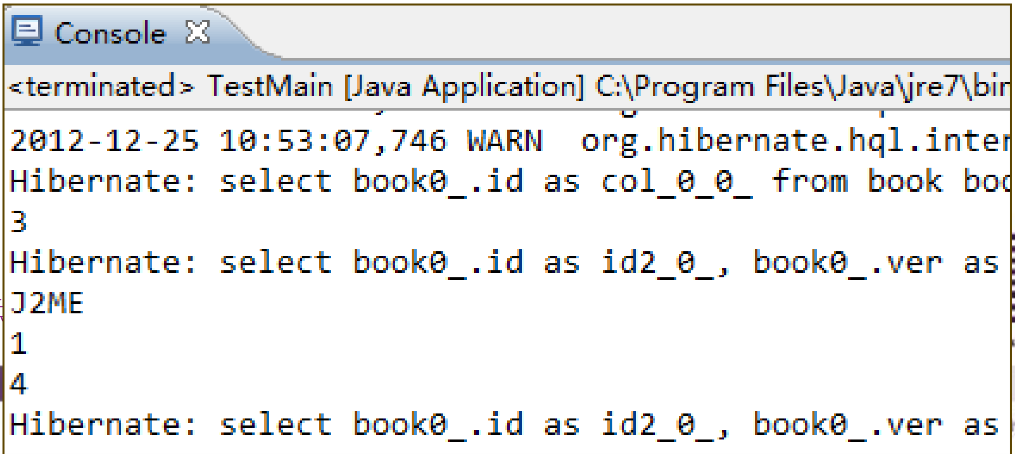
Hibernate Querying

- Execute queries (cont.)
  - Iterates result
    - Occasionally, you might be able to achieve better performance by executing the query using the iterate() method, if you expect that the actual entity instances returned by the query will already be in the session or second-level cache
    - If they are not already cached, iterate()will be slower than list() and might require many database hits for a simple query

Using Iterator Demo



```java
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();
java.util.Iterator books = session
        .createQuery("from Book where id>? and id<? and name like 'j%'")
        .setParameter(0, 1).setParameter(1, 5).iterate();

for (; books.hasNext();) {
    Book b = (Book) books.next();
    System.out.println(b.getId());
    System.out.println(b.getName());
    System.out.println(b.getVer());
}
session.getTransaction().commit();
```



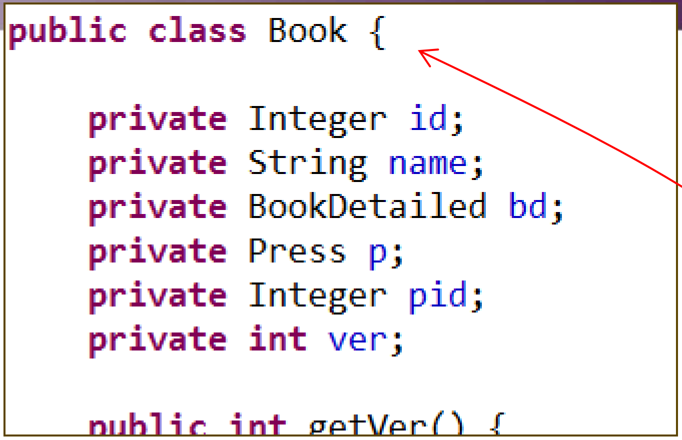
Console ⊠

```
<terminated> TestMain [Java Application] C:\Program Files\Java\jre7\bir
2012-12-25 10:53:07,746 WARN  org.hibernate.hql.inter
Hibernate: select book0_.id as col_0_0_ from book boo
3
Hibernate: select book0_.id as id2_0_, book0_.ver as
J2ME
1
4
Hibernate: select book0_.id as id2_0_, book0_.ver as
```

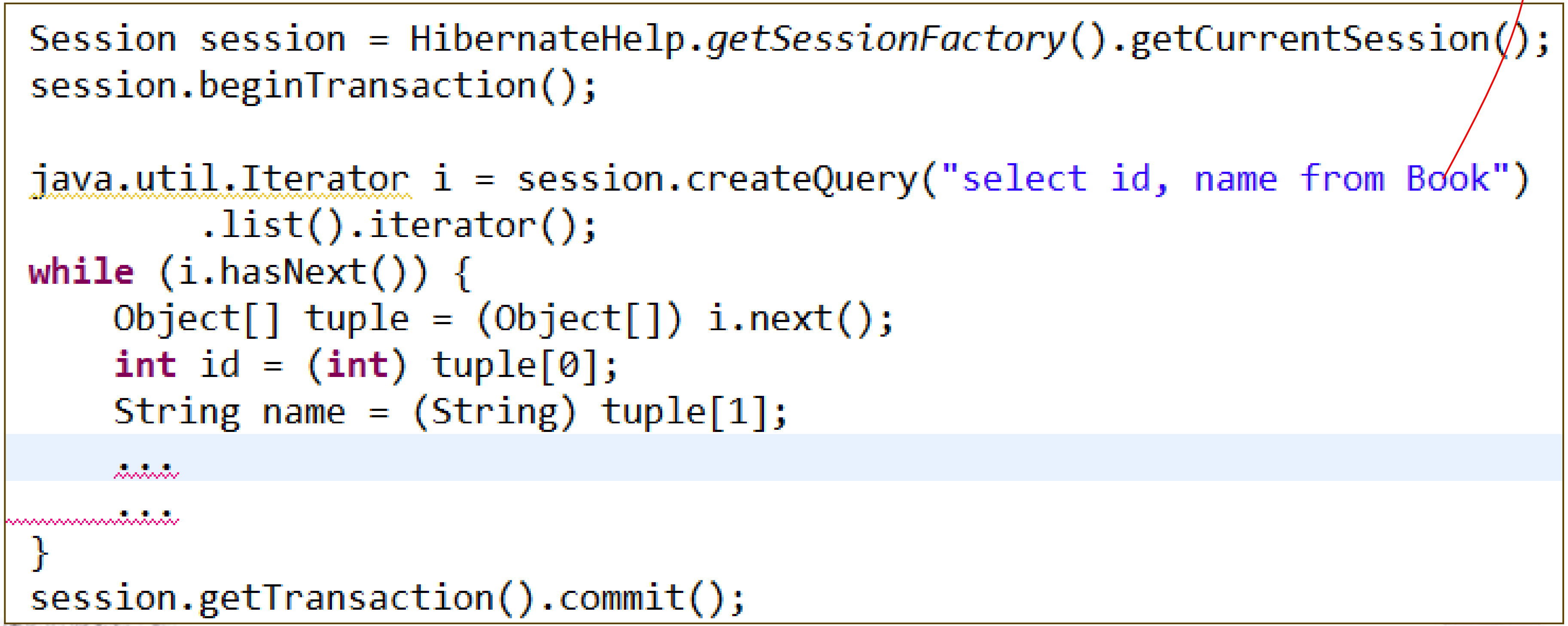
Hibernate Querying

- Execute queries (cont.)
  - Scalar results
    - Queries can specify a property of a class in the select clause.
    - They can even call SQL aggregate functions. Properties or aggregates are considered "scalar" results and not entities in persistent state
  - Queries that return tuples
    - Hibernate queries sometimes return tuples of objects. Each tuple is returned as an array

Scalar results Demo

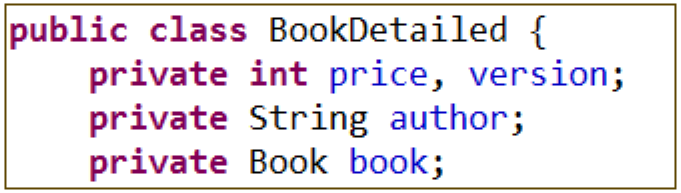


```
public class Book {

    private Integer id;
    private String name;
    private BookDetailed bd;
    private Press p;
    private Integer pid;
    private int ver;

    public int getVer() {
```

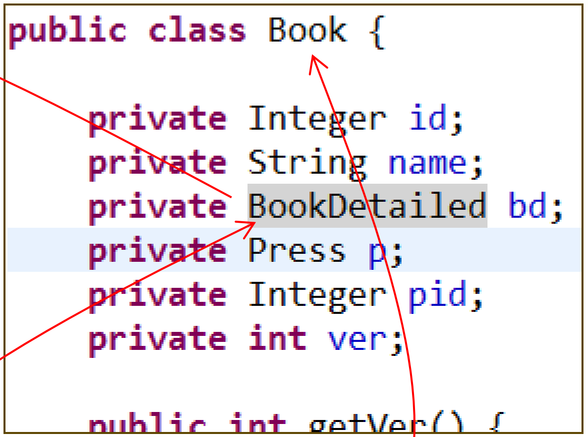


```
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();

java.util.Iterator i = session.createQuery("select id, name from Book")
        .list().iterator();
while (i.hasNext()) {
    Object[] tuple = (Object[]) i.next();
    int id = (int) tuple[0];
    String name = (String) tuple[1];
    ...
    ...
}
session.getTransaction().commit();
```

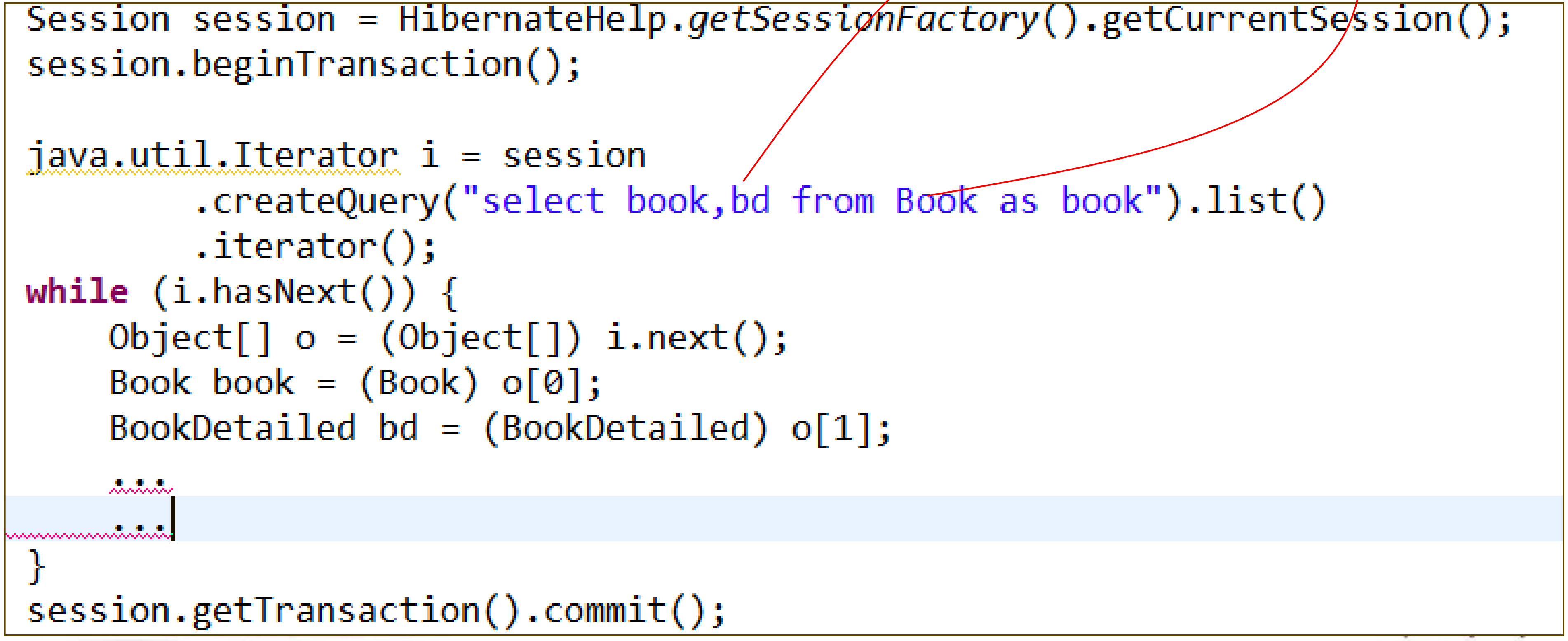
Queries that return tuples Demo



```java
public class BookDetailed {
    private int price, version;
    private String author;
    private Book book;
```



```java
public class Book {

    private Integer id;
    private String name;
    private BookDetailed bd;
    private Press p;
    private Integer pid;
    private int ver;

    public int getVer() {
```

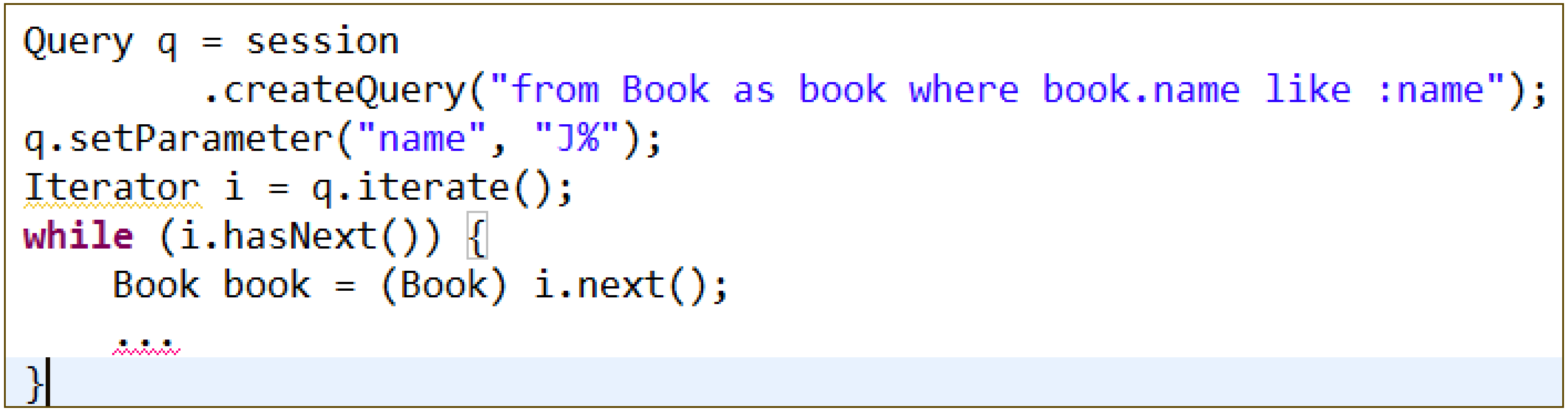


```java
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();

java.util.Iterator i = session
        .createQuery("select book,bd from Book as book").list()
        .iterator();
while (i.hasNext()) {
    Object[] o = (Object[]) i.next();
    Book book = (Book) o[0];
    BookDetailed bd = (BookDetailed) o[1];
    ...
    ...
}
session.getTransaction().commit();
```

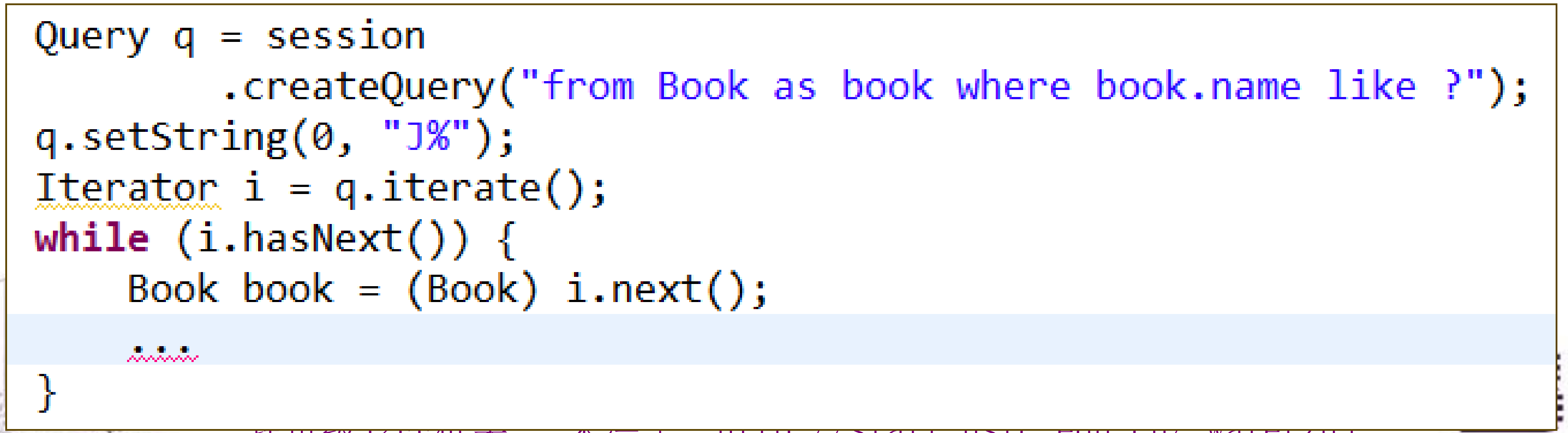
Hibernate Querying

- Execute queries (cont.)
  - Bind parameters
    - Methods on Query are provided for binding values to named parameters or JDBC-style ? Parameters
    - Named parameters are identifiers of the form :paraname in the query string
      - named parameters are insensitive to the order they occur in the query string
      - they can occur multiple times in the same query
      - they are self-documenting

Bind Parameters Demo 1

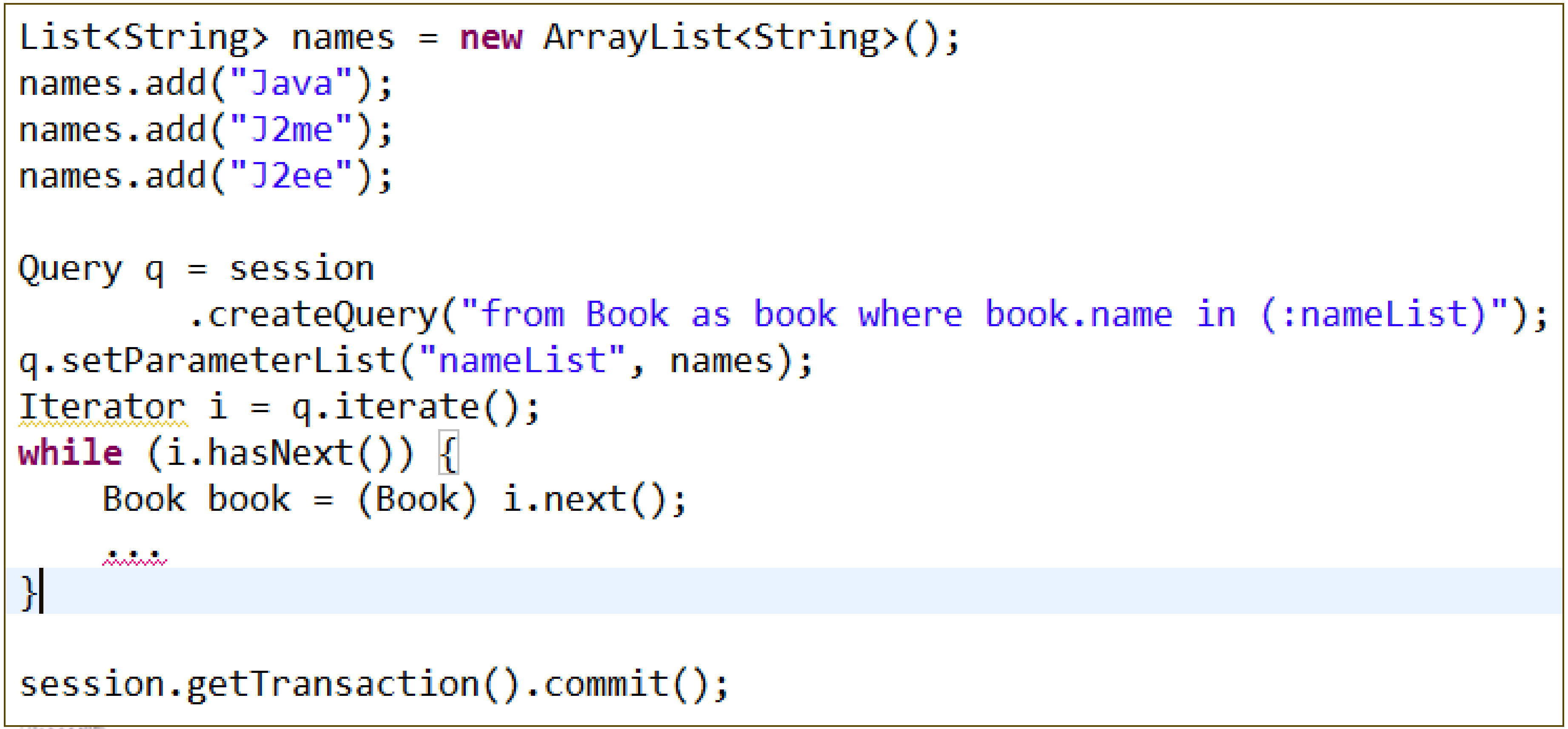


```
Query q = session
        .createQuery("from Book as book where book.name like :name");
q.setParameter("name", "J%");
Iterator i = q.iterate();
while (i.hasNext()) {
    Book book = (Book) i.next();
    ...
}
```



```
Query q = session
        .createQuery("from Book as book where book.name like ?");
q.setString(0, "J%");
Iterator i = q.iterate();
while (i.hasNext()) {
    Book book = (Book) i.next();
    ...
}
```

Bind Parameters Demo 2

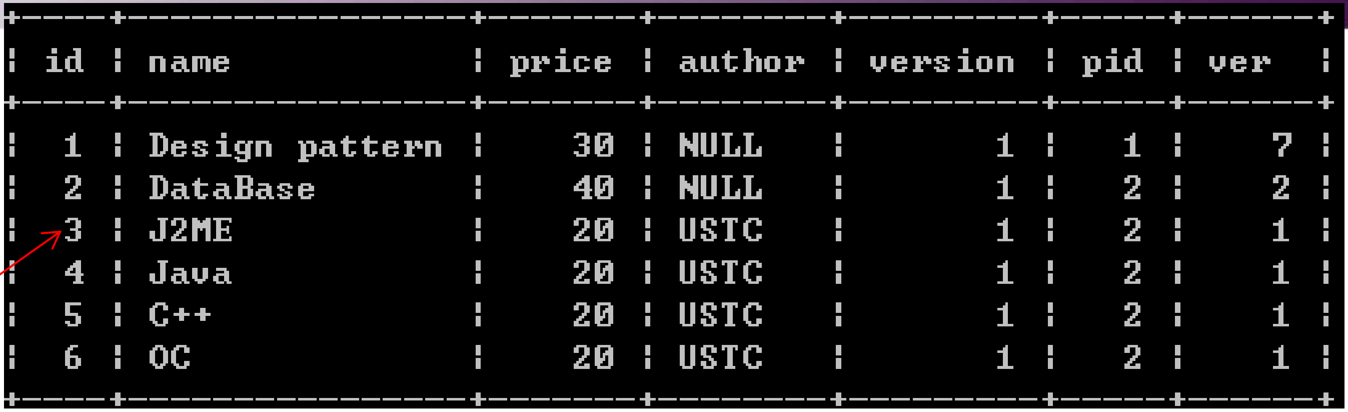


```java
List<String> names = new ArrayList<String>();
names.add("Java");
names.add("J2me");
names.add("J2ee");

Query q = session
        .createQuery("from Book as book where book.name in (:nameList)");
q.setParameterList("nameList", names);
Iterator i = q.iterate();
while (i.hasNext()) {
    Book book = (Book) i.next();
    ...
}

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

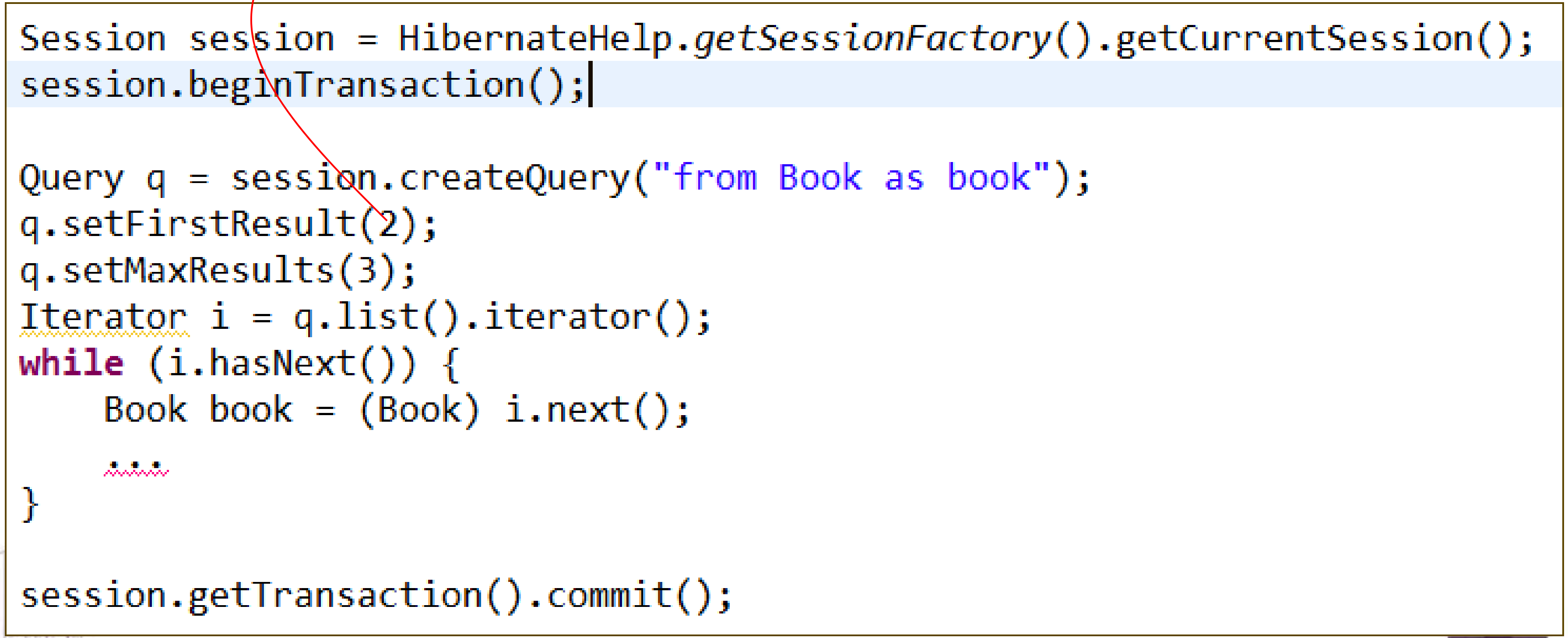
Hibernate Querying

- Execute queries (cont.)
  - Pagination
    - If you need to specify bounds upon your result set, that is, the maximum number of rows you want to retrieve
    - And/or the first row you want to retrieve, you can use methods of the Query interface
    - Hibernate knows how to translate those limit queries into the native SQL of your DBMS

Pagination Demo



```
+----+------------------+-------+--------+---------+-----+-----+
| id | name             | price | author | version | pid | ver |
+----+------------------+-------+--------+---------+-----+-----+
|  1 | Design pattern   |    30 | NULL   |       1 |   1 |   7 |
|  2 | DataBase         |    40 | NULL   |       1 |   2 |   2 |
|  3 | J2ME             |    20 | USTC   |       1 |   2 |   1 |
|  4 | Java             |    20 | USTC   |       1 |   2 |   1 |
|  5 | C++              |    20 | USTC   |       1 |   2 |   1 |
|  6 | OC               |    20 | USTC   |       1 |   2 |   1 |
+----+------------------+-------+--------+---------+-----+-----+
```



```java
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();

Query q = session.createQuery("from Book as book");
q.setFirstResult(2);
q.setMaxResults(3);
Iterator i = q.list().iterator();
while (i.hasNext()) {
    Book book = (Book) i.next();
    ...
}

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

Hibernate Querying

- HQL (Hibernate Query Language)
  - Hibernate uses a powerful query language (HQL) that is similar in appearance to SQL
  - Compared with SQL, however, HQL is fully object-oriented and understands notions like inheritance, polymorphism and association
  - HQL queries are case-insensitive

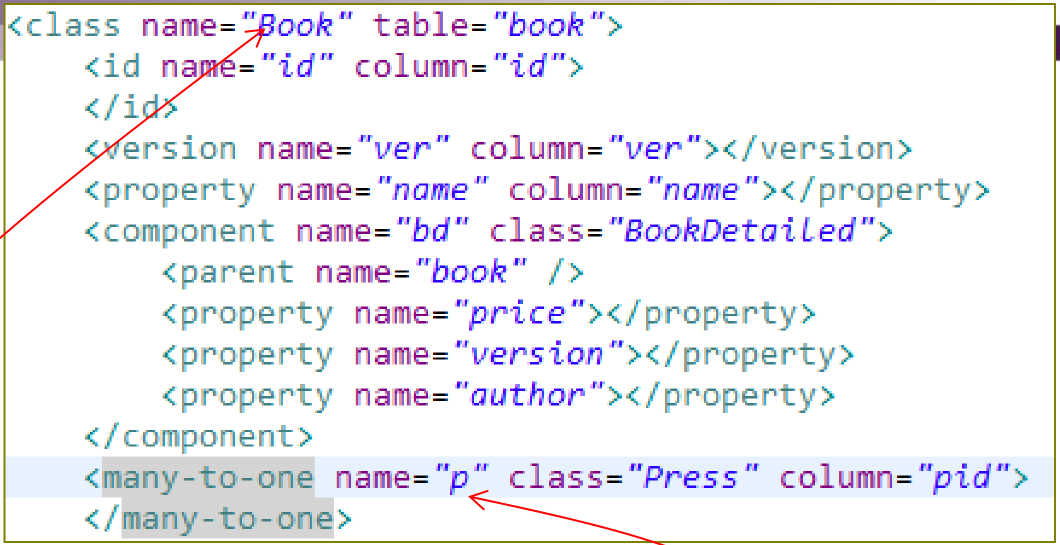
Hibernate Querying

- HQL (cont.)
  - From clause
    - from ClassName
      - This returns all instances of the class
    - from ClassName [as] alias
      - You can assign an alias to Class, and use it later in query
      - as keyword is optional
    - from Class1 [as] [alias1], Class2 [as] [alias2],…
      - Multi classes can be separated by comma

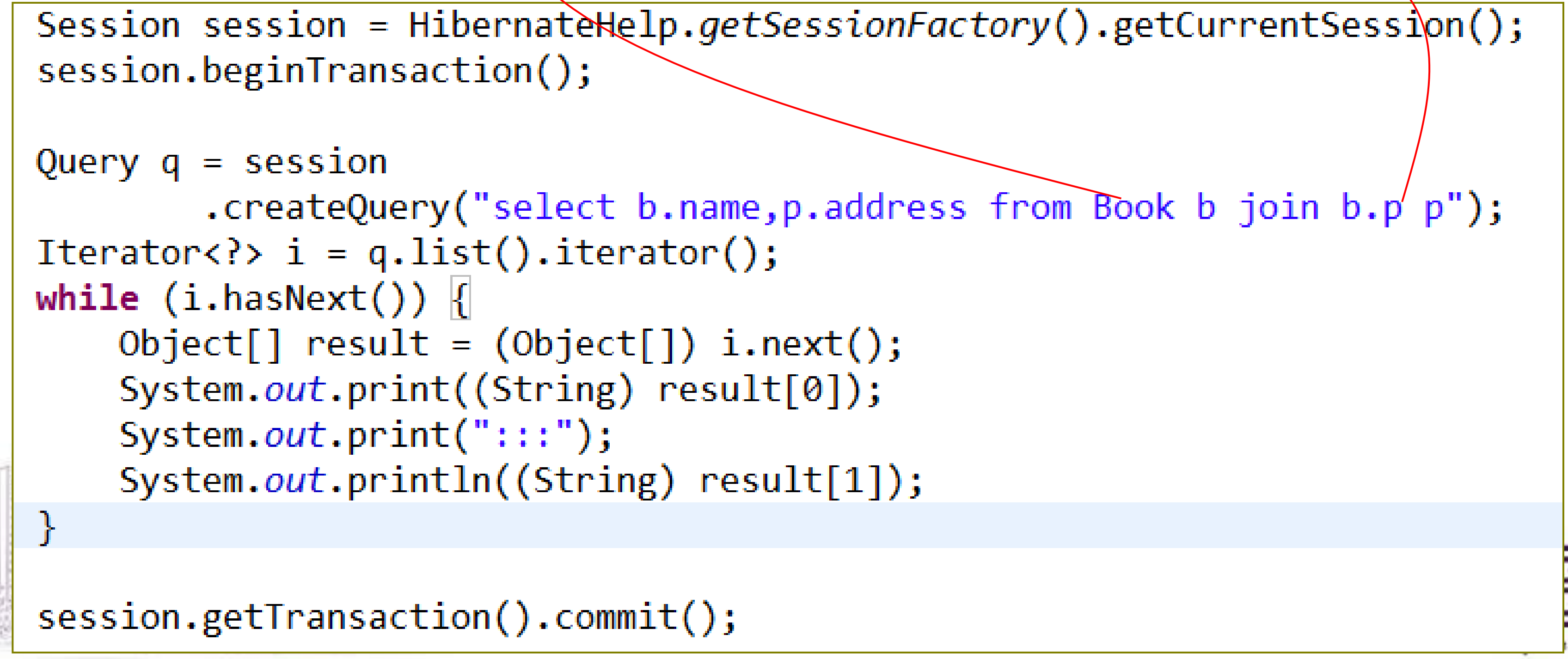
Hibernate Querying

- HQL (cont.)
  - Associations and joins
    - You can also assign aliases to associated entities or to elements of a collection of values using a join
    - The supported join types are borrowed from ANSI SQL
      - Inner join
      - Left outer join
      - Right outer join
      - Fully outer join

Querying Many to One Association by Using Inner Join Demo



```xml
<class name="Book" table="book">
    <id name="id" column="id">
    </id>
    <version name="ver" column="ver"></version>
    <property name="name" column="name"></property>
    <component name="bd" class="BookDetailed">
        <parent name="book" />
        <property name="price"></property>
        <property name="version"></property>
        <property name="author"></property>
    </component>
    <many-to-one name="p" class="Press" column="pid">
    </many-to-one>
```



```java
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();

Query q = session
        .createQuery("select b.name,p.address from Book b join b.p p");
Iterator<?> i = q.list().iterator();
while (i.hasNext()) {
    Object[] result = (Object[]) i.next();
    System.out.print((String) result[0]);
    System.out.print(":::");
    System.out.println((String) result[1]);
}

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

Hibernate Querying

- HQL (cont.)
  - Associations and joins (cont.)
    - You may supply extra join conditions using the HQL with keyword
    - Forms of join syntax
      - Explicit
      - Implicit
    - implicit join result in inner joins in the resulting SQL statement

Implicit and Explicit Join Demo

With “with” key word for join conditions

```
from Cat as cat
    left join cat.kittens as kitten
        with kitten.bodyWeight > 10.0
```

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

```
from Cat as cat where cat.mate.name like '%s%'
```

Implicit join between cat and mate

Hibernate Querying

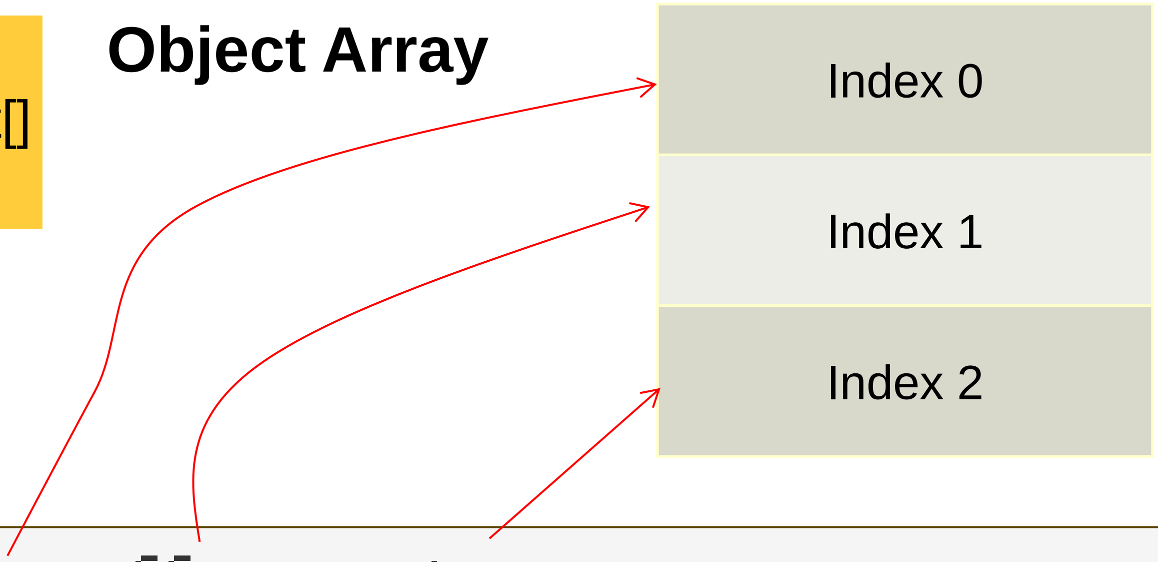
- HQL (cont.)
  - Referring to identifier property
    - The special property (lowercase) "id" may be used to reference the identifier property of an entity *provided that the entity does not define a non-identifier property named id*
    - If the entity defines a named identifier property, you can use that property name
    - References to composite identifier properties follow the same naming rules

Hibernate Querying

- HQL (cont.)
  - Select clause
    - The select clause picks which objects and properties to return in the query result set
    - Queries can return properties of any value type including properties of component type
    - Queries can return multiple objects and/or properties as an array of type Object[] or as a list
    - You can assign aliases to selected expressions using as

Queries Return an array of Type Object[] Demo

Object Array



```
select mother, offspr, mate.name
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

Hibernate Querying

- HQL (cont.)
  - Where clause
    - The where clause allows you to refine the list of instances returned. If no alias exists, you can refer to properties by name
    - The = operator can be used to compare not only properties, but also instances
    - The special property (lowercase) "id" can be used to reference the unique identifier of an object

Hibernate Querying

- HQL (cont.)
  - Aggregate functions
    - HQL queries can even return the results of aggregate functions on properties
      - avg(), sum(), min(), max(), count(), count(distinct), count(all), +-*/, etc
  - Polymorphic queries
    - Hibernate queries can name *any*Java class or interface in the from clause. The query will return instances of all persistent classes that extend that class or implement the interface

Hibernate Querying

- HQL (cont.)
  - Expressions
    - Expressions used in the where clause include the following
      - mathematical operators
      - binary comparison operators
      - logical operations
      - parentheses ( ) that indicates grouping
      - in, not in, between, is null, etc.
      - Etc.

Expressions in Where Clause Demo

```
from DomesticCat cat where cat.name between 'A' and 'B'

from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

```
select cust
from Product prod,
    Store store
    inner join store.customers cust
where prod.name = 'widget'
    and store.location.name in ( 'Melbourne', 'Sydney' )
    and prod = all elements(cust.currentOrder.lineItems)
```

Hibernate Querying

- HQL (cont.)
  - Order by clause
    - The list returned by a query can be ordered by any property of a returned class or components
  - Group by clause
    - A query that returns aggregate values can be grouped by any property of a returned class or components
    - A having clause is also allowed

Order by and Group by Clause Demo

```
from DomesticCat cat
order by cat.name asc, cat.weight desc, cat.birthdate
```

```
select cat
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.name, cat.other, cat.properties
having avg(kitten.weight) > 100
order by count(kitten) asc, sum(kitten.weight) desc
```

Hibernate Querying

- HQL (cont.)
  - Subqueries
    - For databases that support subselects, Hibernate supports subqueries within queries
    - A subquery must be surrounded by parentheses (often by an SQL aggregate function call).
    - Even correlated subqueries (subqueries that refer to an alias in the outer query) are allowed
    - HQL subqueries can occur only in the select or where clauses

Subqueries Demo

```
from Cat as fatcat
where fatcat.weight > (
    select avg(cat.weight) from DomesticCat cat
)
```

```
select cat.id, (select max(kit.weight) from cat.kitten kit)
from Cat as cat
```

```
from DomesticCat as cat
where cat.name not in (
    select name.nickName from Name as name
)
```

Hibernate Querying

- Criteria Queries (QBC)
  - The interface org.hibernate.Criteria represents a query against a particular persistent class. The Session is a factory for Criteria instances
  - An individual query criterion is an instance of the interface org.hibernate.criterion.Criterion
  - The class org.hibernate.criterion.Restrictions defines factory methods for obtaining certain built-in Criterion types

Hibernate Querying

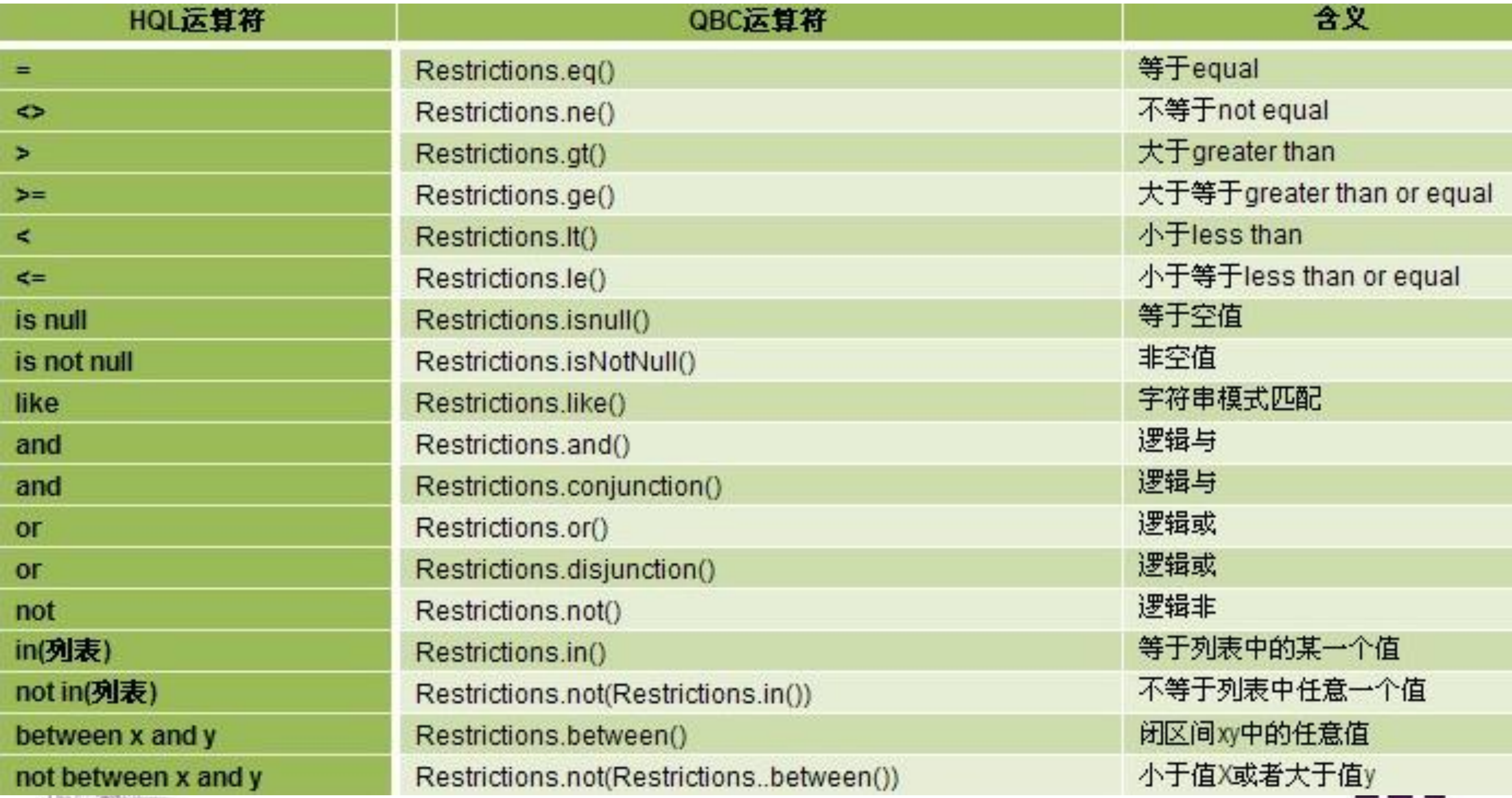
- QBC (cont.)
  - Narrowing the result set
    - Restrictions can be grouped logically
    - There are a range of built-in criterion types (Restrictions subclasses)
    - You can also obtain a criterion from a Property instance. You can create a Property by calling Property.forName()

Narrowing the result set

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.in( "name", new String[] { "Fritz", "Izi", "Pk" } ) )
    .add( Restrictions.disjunction()
        .add( Restrictions.isNull("age") )
        .add( Restrictions.eq("age", new Integer(0) ) )
        .add( Restrictions.eq("age", new Integer(1) ) )
        .add( Restrictions.eq("age", new Integer(2) ) )
    ) )
    .list();
```

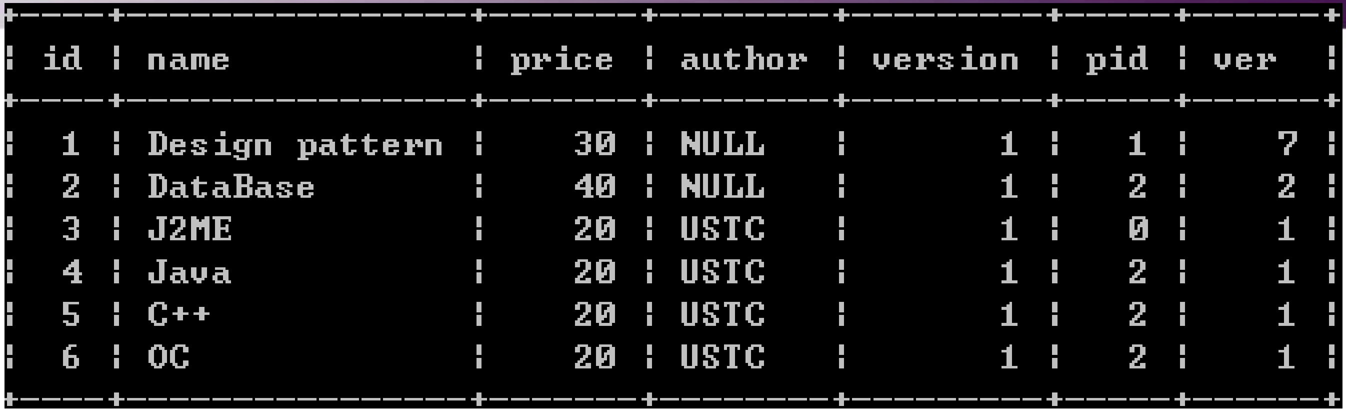
```
Property age = Property.forName("age");
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.disjunction()
        .add( age.isNull() )
        .add( age.eq( new Integer(0) ) )
        .add( age.eq( new Integer(1) ) )
        .add( age.eq( new Integer(2) ) )
    ) )
    .add( Property.forName("name").in( new String[] { "Fritz", "Izi", "Pk" } ) )
    .list();
```

QBC Operators

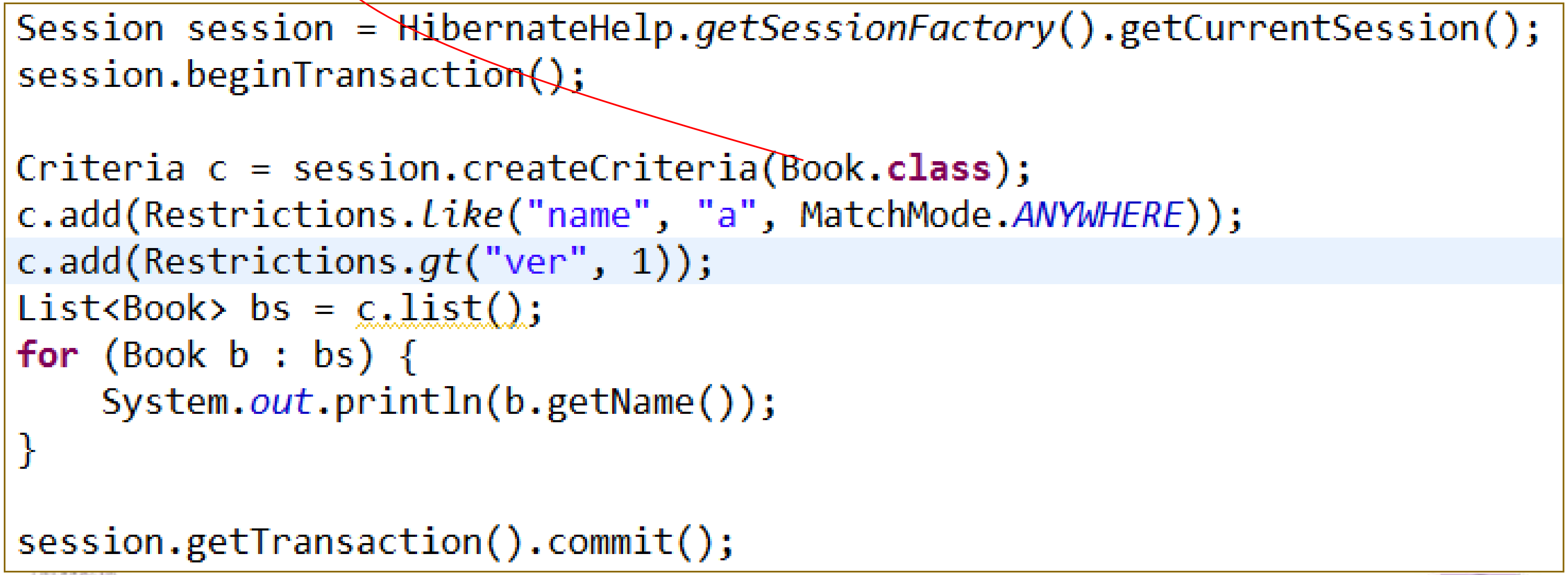


HQL运算符	QBC运算符	含义
=	Restrictions.eq()	等于equal
<>	Restrictions.ne()	不等于not equal
>	Restrictions.gt()	大于greater than
>=	Restrictions.ge()	大于等于greater than or equal
<	Restrictions.lt()	小于less than
<=	Restrictions.le()	小于等于less than or equal
is null	Restrictions.isnull()	等于空值
is not null	Restrictions.isNotNull()	非空值
like	Restrictions.like()	字符串模式匹配
and	Restrictions.and()	逻辑与
and	Restrictions.conjunction()	逻辑与
or	Restrictions.or()	逻辑或
or	Restrictions.disjunction()	逻辑或
not	Restrictions.not()	逻辑非
in(列表)	Restrictions.in()	等于列表中的某一个值
not in(列表)	Restrictions.not(Restrictions.in())	不等于列表中任意一个值
between x and y	Restrictions.between()	闭区间xy中的任意值
not between x and y	Restrictions.not(Restrictions..between())	小于值X或者大于值y

QBC Demo



```
+----+------------------+--------+----------+-----------+------+-------+
| id | name             | price  | author   | version   | pid  | ver   |
+----+------------------+--------+----------+-----------+------+-------+
|  1 | Design pattern   |     30 | NULL     |         1 |    1 |     7 |
|  2 | DataBase         |     40 | NULL     |         1 |    2 |     2 |
|  3 | J2ME             |     20 | USTC     |         1 |    0 |     1 |
|  4 | Java             |     20 | USTC     |         1 |    2 |     1 |
|  5 | C++              |     20 | USTC     |         1 |    2 |     1 |
|  6 | OC               |     20 | USTC     |         1 |    2 |     1 |
+----+------------------+--------+----------+-----------+------+-------+
```



```java
Session session = HibernateHelp.getSessionFactory().getCurrentSession();
session.beginTransaction();

Criteria c = session.createCriteria(Book.class);
c.add(Restrictions.like("name", "a", MatchMode.ANYWHERE));
c.add(Restrictions.gt("ver", 1));
List<Book> bs = c.list();
for (Book b : bs) {
    System.out.println(b.getName());
}

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

Hibernate Querying

- QBC (cont.)
  - Ordering result
    - You can order the results using org.hibernate.criterion.Order
  - Associations
    - By navigating associations using createCriteria() you can specify constraints upon related entities
  - Dynamic association fetching
    - You can specify association fetching semantics at runtime using setFetchMode()

Ordering Result and Associations Demo

```
List cats = sess.createCriteria(Cat.class)
    .add( Property.forName("name").like("F%") )
    .addOrder( Property.forName("name").asc() )
    .addOrder( Property.forName("age").desc() )
    .setMaxResults(50)
    .list();
```

```
List cats = sess.createCriteria(Cat.class)
    .createAlias("kittens", "kt")
    .createAlias("mate", "mt")
    .add( Restrictions.eqProperty("kt.name", "mt.name") )
    .list();
```

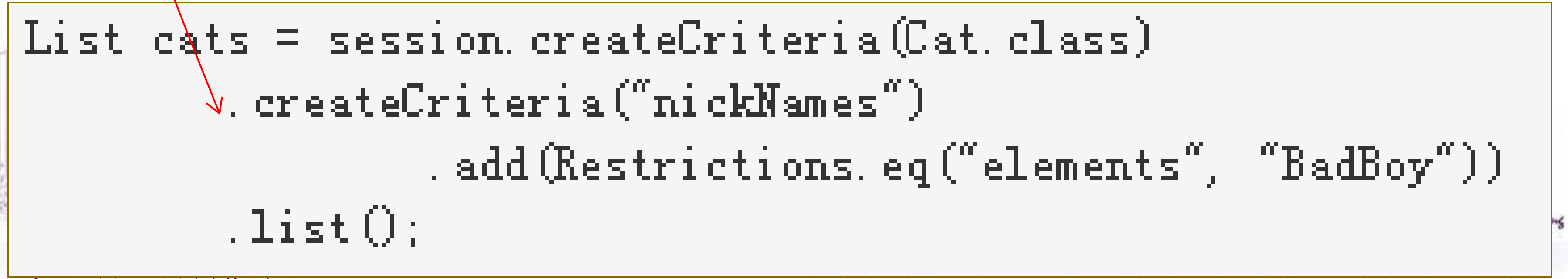
Hibernate Querying

- QBC (cont.)
  - Components
    - To add a restriction against a property of an embedded component, the component property name should be prepended to the property name when creating the Restriction
    - The criteria object should be created on the owning entity, and cannot be created on the component itself

```
List cats = session.createCriteria(Cat.class)
        .add(Restrictions.eq("fullName.lastName", "Cattington"))
        .list();
```

Hibernate Querying

- QBC (cont.)
  - Collections
    - When using criteria against collections, there are two distinct cases
      - One is if the collection contains entities (eg. <one-to-many/> or <many-to-many/>) or components (<composite-element/>), **using criteria as association**
      - The second is if the collection contains scalar values (<element/>)



```
List cats = session.createCriteria(Cat.class)
        .createCriteria("nickNames")
                .add(Restrictions.eq("elements", "BadBoy"))
        .list();
```

Hibernate Querying

- QBC (cont.)
  - Projections
    - The class org.hibernate.criterion.Projections is a factory for Projection instances. You can apply a projection to a query by calling setProjection()
  - Grouping
    - There is no explicit "group by" necessary in a criteria query. Certain projection types are defined to be *grouping projections*, which also appear in the SQL group by clause

Projections and Grouping Demo

```
List results = session.createCriteria(Cat.class)
      .setProjection( Projections.rowCount() )
      .add( Restrictions.eq("color", Color.BLACK) )
      .list();
```

```
List results = session.createCriteria(Cat.class)
     .setProjection( Projections.projectionList()
          .add( Projections.rowCount(), "catCountByColor" )
          .add( Projections.avg("weight"), "avgWeight" )
          .add( Projections.max("weight"), "maxWeight" )
          .add( Projections.groupProperty("color"), "color" )
     )
     .addOrder( Order.desc("catCountByColor") )
     .addOrder( Order.desc("avgWeight") )
     .list();
```

Hibernate Querying

- QBC (cont.)
  - Detached queries
    - The DetachedCriteria class allows you to create a query outside the scope of a session and then execute it using an arbitrary Session
  - Subqueries
    - A DetachedCriteria can also be used to express a subquery. Criterion instances involving subqueries can be obtained via Subqueries or Property

Detached Queries and Subqueries Demo

```
DetachedCriteria query = DetachedCriteria.forClass(Cat.class)
    .add( Property.forName("sex").eq('F') );

Session session = ....;
Transaction txn = session.beginTransaction();
List results = query.getExecutableCriteria(session).setMaxResults(100).list();
txn.commit();
session.close();
```

```
DetachedCriteria avgWeightForSex = DetachedCriteria.forClass(Cat.class, "cat2")
    .setProjection( Property.forName("weight").avg() )
    .add( Property.forName("cat2.sex").eqProperty("cat.sex") );
session.createCriteria(Cat.class, "cat")
    .add( Property.forName("weight").gt(avgWeightForSex) )
    .list();
```

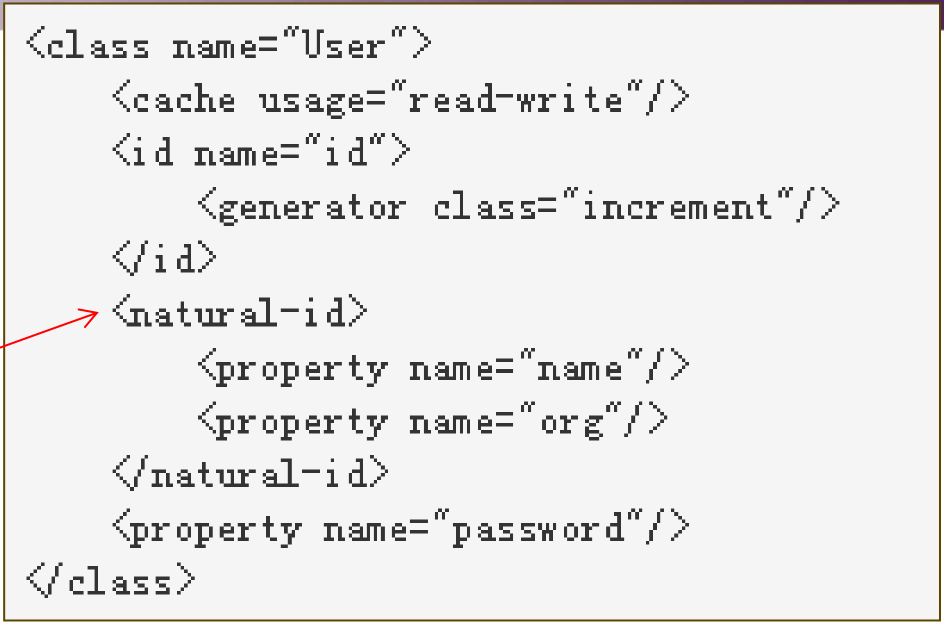
Hibernate Querying

- QBC (cont.)
  - Queries by natural identifier
    - For most queries, including criteria queries, the query cache is not efficient because query cache invalidation occurs too frequently
    - However, there is a special kind of query where you can optimize the cache invalidation algorithm: lookups by a constant natural key
    - The criteria API provides special provision for this use case

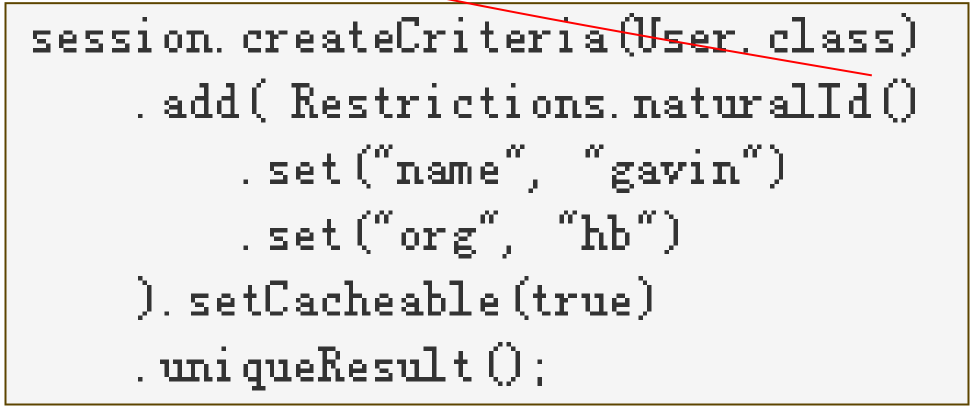
Hibernate Querying

- QBC (cont.)
  - Queries by natural identifier
    - First, map the natural key of your entity using <natural-id> and enable use of the second-level cache
    - Once you have enabled the Hibernate query cache, the Restrictions.naturalId() allows you to make use of the more efficient cache algorithm

Queries By Natural Identifier Demo



```
<class name="User">
    <cache usage="read-write"/>
    <id name="id">
        <generator class="increment"/>
    </id>
    <natural-id>
        <property name="name"/>
        <property name="org"/>
    </natural-id>
    <property name="password"/>
</class>
```



```
session.createCriteria(User.class)
    .add( Restrictions.naturalId()
        .set("name", "gavin")
        .set("org", "hb")
    ).setCacheable(true)
    .uniqueResult();
```

Hibernate Querying

- Example Queries (QBE)
  - The class org.hibernate.criterion.Example allows you to construct a query criterion from a given instance
  - Version properties, identifiers and associations are ignored. By default, null valued properties are excluded
  - You can even use examples to place criteria upon associated objects

QBE Demo

```
Cat cat = new Cat();
cat.setSex('F');
cat.setColor(Color.BLACK);
List results = session.createCriteria(Cat.class)
     .add( Example.create(cat) )
     .list();
```

```
List results = session.createCriteria(Cat.class)
     .add( Example.create(cat) )
     .createCriteria("mate")
          .add( Example.create( cat.getMate() ) )
     .list();
```

Hibernate Querying

- Native SQL
  - You can also express queries in the native SQL dialect of your database
  - Hibernate3 allows you to specify handwritten SQL, including stored procedures, for all create, update, delete, and load operations
  - The most basic SQL query is to get a list of scalars (values)

Native SQL Demo

```
sess.createSQLQuery("SELECT * FROM CATS").list();
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").list();
```

```
sess.createSQLQuery("SELECT * FROM CATS")
 .addScalar("ID", Hibernate.LONG)
 .addScalar("NAME", Hibernate.STRING)
 .addScalar("BIRTHDATE", Hibernate.DATE)
```

```
sess.createSQLQuery("SELECT * FROM CATS").addEntity(Cat.class);
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").addEntity(Cat.class);
```

Hibernate Querying

- Named queries
  - Put all HQL into the XML mapping file or via annotation
  - Be supported in both HQL or native SQL
  - You can place a named query inside ‘hibernate-mapping‘ element, but do not put before the ‘class‘ element
    - Hibernate will prompt invalid mapping file, all your named queries have to put after the ‘class‘ element

Named Queries Demo

```
List people = sess.getNamedQuery("persons")
    .setString("namePattern", namePattern)
    .setMaxResults(50)
    .list();
```

```
<sql-query name="persons">
    <return alias="person" class="eg.Person"/>
    SELECT person.NAME AS {person.name},
           person.AGE AS {person.age},
           person.SEX AS {person.sex}
    FROM PERSON person
    WHERE person.NAME LIKE :namePattern
</sql-query>
```

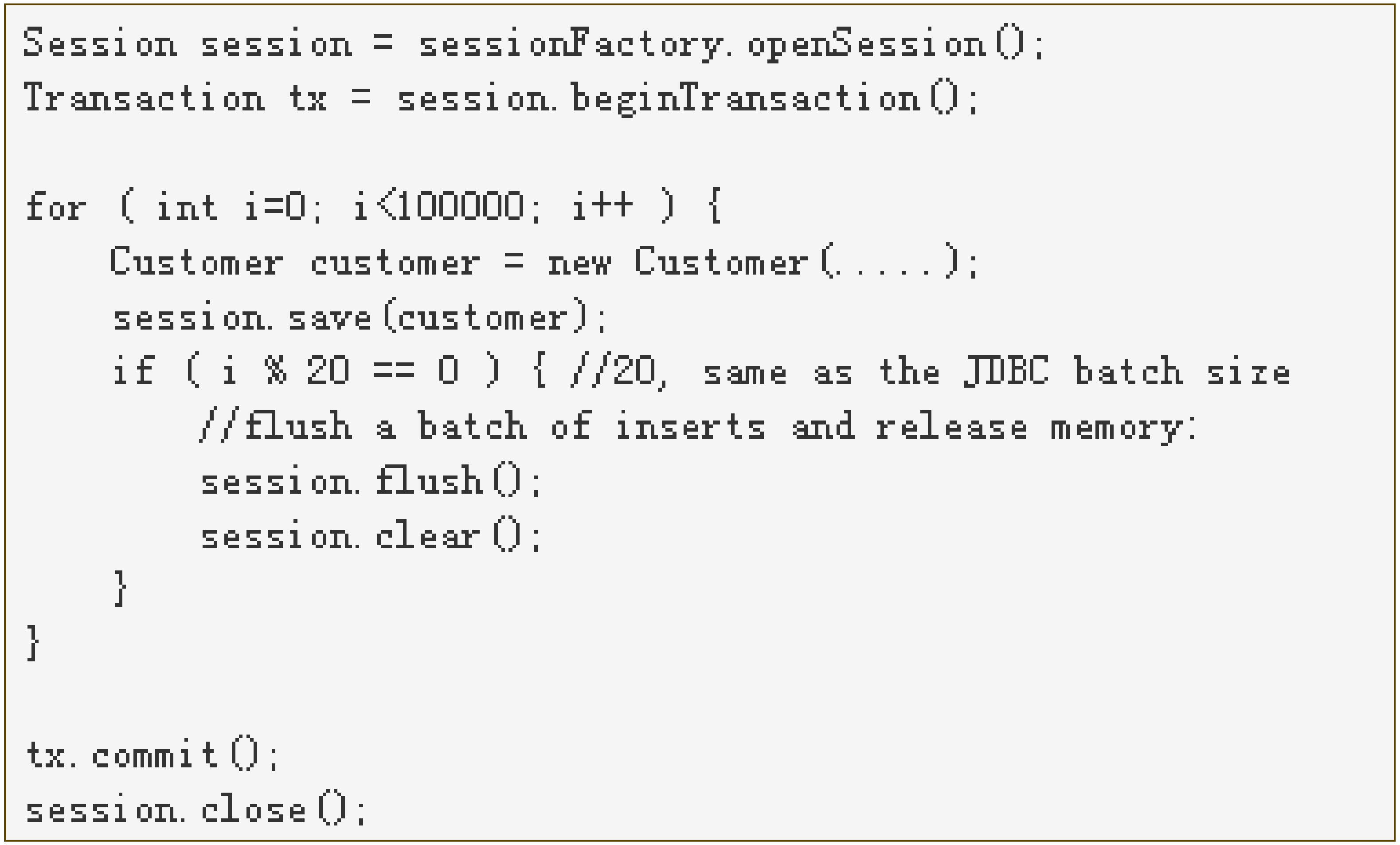
Batch Processing

- Because Hibernate caches all the newly inserted Customer instances in the session-level cache
- A naive approach to inserting 100,000 rows into the database will fall over with an OutOfMemoryException
- If you are undertaking batch processing you will need to enable the use of JDBC batching

Batch Processing

- Batch Inserts
  - When making new objects persistent flush() and then clear() the session regularly in order to control the size of the first-level cache
- Batch Updates
  - For retrieving and updating data, the same ideas apply as batch inserts
  - In addition, you need to use scroll() to take advantage of server-side cursors for queries that return many rows of data

Batch Inserts Demo



```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
    if ( i % 20 == 0 ) { //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

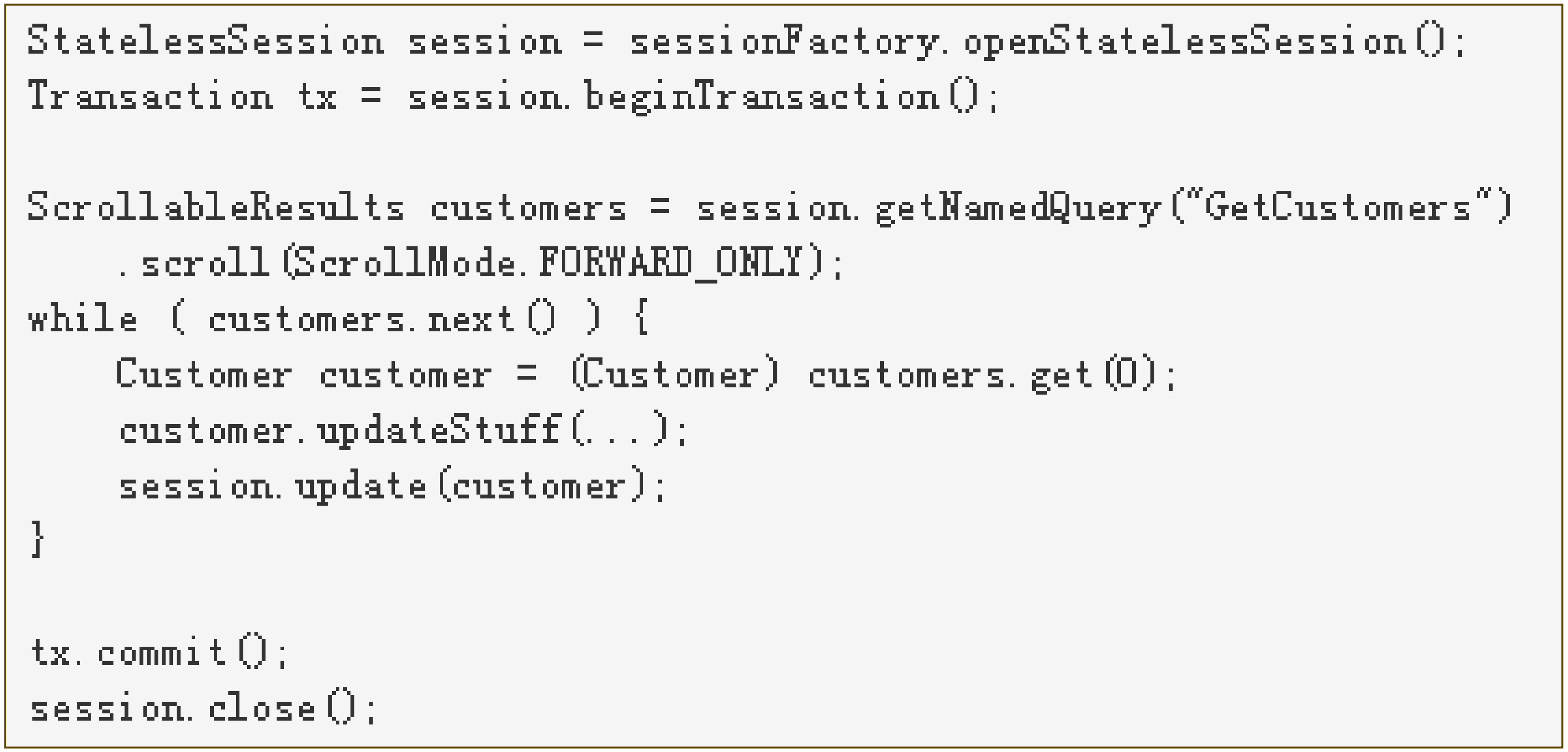
Batch Updates Demo

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")
    .setCacheMode(CacheMode.IGNORE)
    .scroll(ScrollMode.FORWARD_ONLY);
int count=0;
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    if ( ++count % 20 == 0 ) {
        //flush a batch of updates and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

Batch Processing

- The StatelessSession Interface
  - Alternatively, Hibernate provides a command-oriented API that can be used for streaming data to and from the database in the form of detached objects
  - A StatelessSession has no persistence context associated with it and does not provide many of the higher-level life cycle semantics

Using StatelessSession Interface Demo



```
StatelessSession session = sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")
    .scroll(ScrollMode.FORWARD_ONLY);
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    session.update(customer);
}

tx.commit();
session.close();
```

Data Filtering

- Hibernate3 provides an innovative new approach to handling data with "visibility" rules. A *Hibernate filter* is a global, named, parameterized filter that can be enabled or disabled for a particular Hibernate session
- Hibernate3 has the ability to pre-define filter criteria and attach those filters at both a class level and a collection level

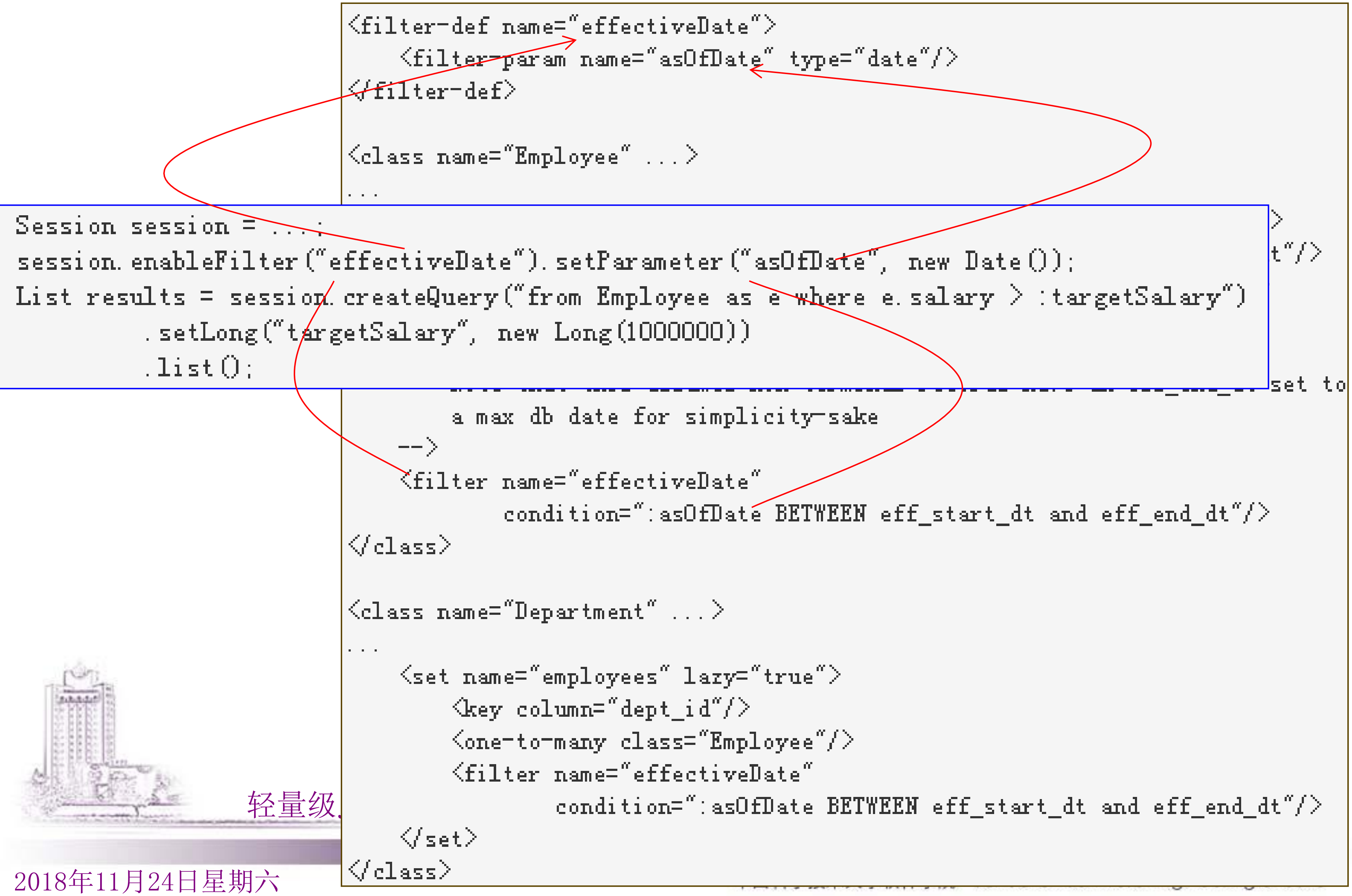
Data Filtering

- A filter criteria allows you to define a restriction clause similar to the existing "where" attribute available on the class and various collection elements. These filter conditions, however, can be parameterized
- The application can then decide at runtime whether certain filters should be enabled and what their parameter values should be

Data Filtering

- Using Hibernate mapping files for defining filters
  - The filters must first be defined and then attached to the appropriate mapping elements
  - To define a filter, use the <filter-def/>element within a <hibernate-mapping/> element
  - Predefined filters can then be attached to a class or collection

Data Filtering Demo



```
<filter-def name="effectiveDate">
    <filter-param name="asOfDate" type="date"/>
</filter-def>

<class name="Employee" ...>
...
                                                                                                  >
                                                                                                 t"/>

                                                                                                 set to
        a max db date for simplicity-sake
    -->
    <filter name="effectiveDate"
            condition=":asOfDate BETWEEN eff_start_dt and eff_end_dt"/>
</class>

<class name="Department" ...>
...
    <set name="employees" lazy="true">
        <key column="dept_id"/>
        <one-to-many class="Employee"/>
        <filter name="effectiveDate"
                condition=":asOfDate BETWEEN eff_start_dt and eff_end_dt"/>
    </set>
</class>
```

```
Session session = ...;
session.enableFilter("effectiveDate").setParameter("asOfDate", new Date());
List results = session.createQuery("from Employee as e where e.salary > :targetSalary")
          .setLong("targetSalary", new Long(1000000))
          .list();
```

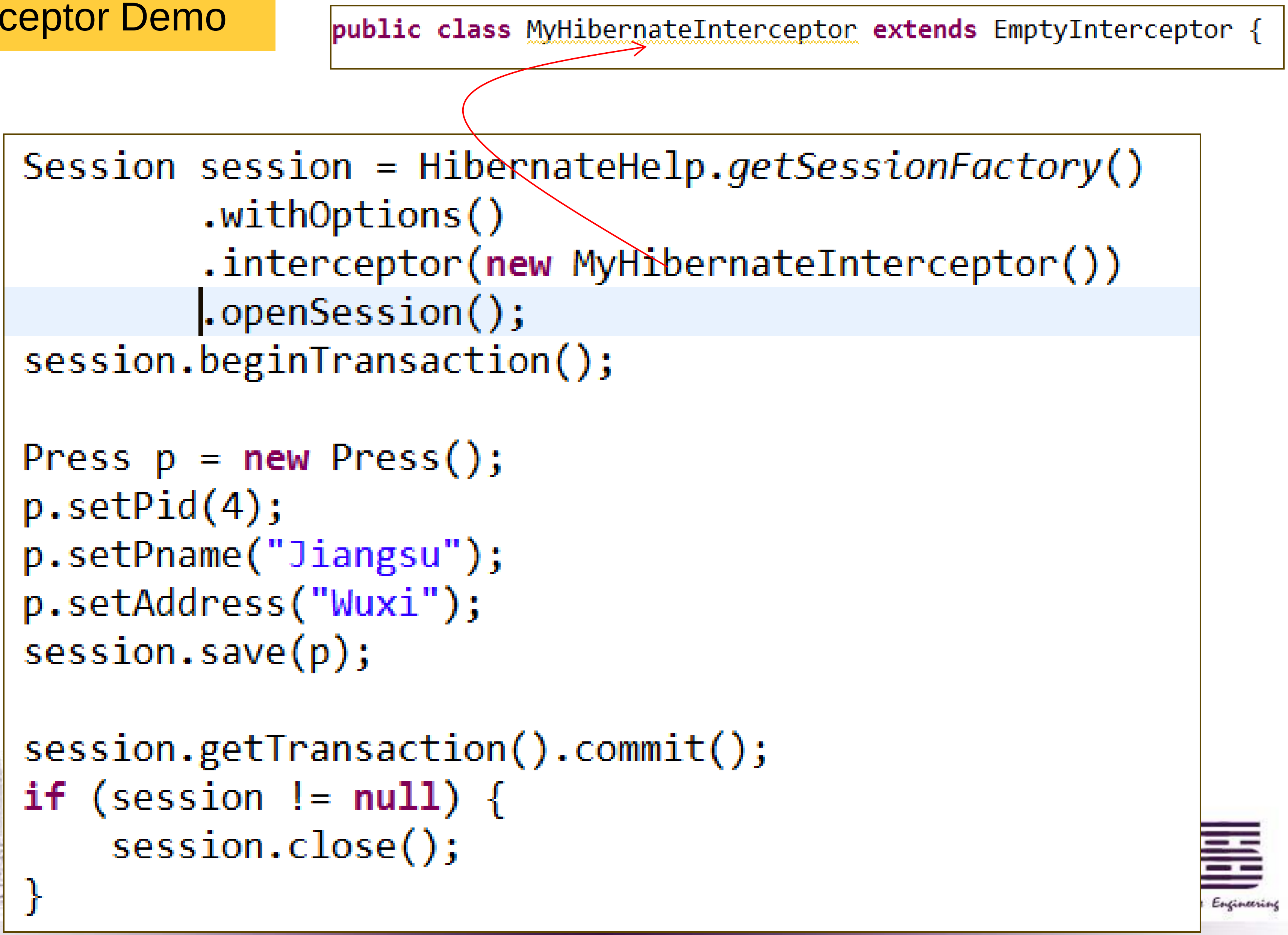
Interceptors and Events

- Interceptors
  - It is useful for the application to react to certain events that occur inside Hibernate
  - The Interceptor interface provides callbacks from the session to the application, allowing the application to inspect and/or manipulate properties of a persistent object before it is saved, updated, deleted or loaded

Interceptors and Events

- Interceptors
  - There are two kinds of inteceptors
    - Session-scoped
      - A Session-scoped interceptor is specified when a session is opened
    - SessionFactory-scoped
      - A SessionFactory-scoped interceptor is registered with the Configuration object prior to building theSessionFactory
      - SessionFactory-scoped interceptors must be thread safe

Session-scoped Interceptor Demo



```java
public class MyHibernateInterceptor extends EmptyInterceptor {
```

```java
Session session = HibernateHelp.getSessionFactory()
        .withOptions()
        .interceptor(new MyHibernateInterceptor())
        .openSession();
session.beginTransaction();

Press p = new Press();
p.setPid(4);
p.setPname("Jiangsu");
p.setAddress("Wuxi");
session.save(p);

session.getTransaction().commit();
if (session != null) {
    session.close();
}
```

Interceptors and Events

- Event System
  - If you have to react to particular events in your persistence layer, you can also use the Hibernate *event* architecture
  - The event system can be used in addition, or as a replacement, for interceptors
  - Many methods of the Session interface correlate to an event type

Conclusions

- Transaction Management and Concurrency
- Hibernate Querying
- Batch Processing
- Data Filtering
- Interceptors and Events