

Object Persistence using Hibernate

An Object-Relational mapping framework
for object persistence

■ Java Object/Relational Mapping

- Open source – LGPL
- <http://www.hibernate.org/> (JBoss Group)

■ Lets you avoid SQL

■ Works with (almost) all relational DBs

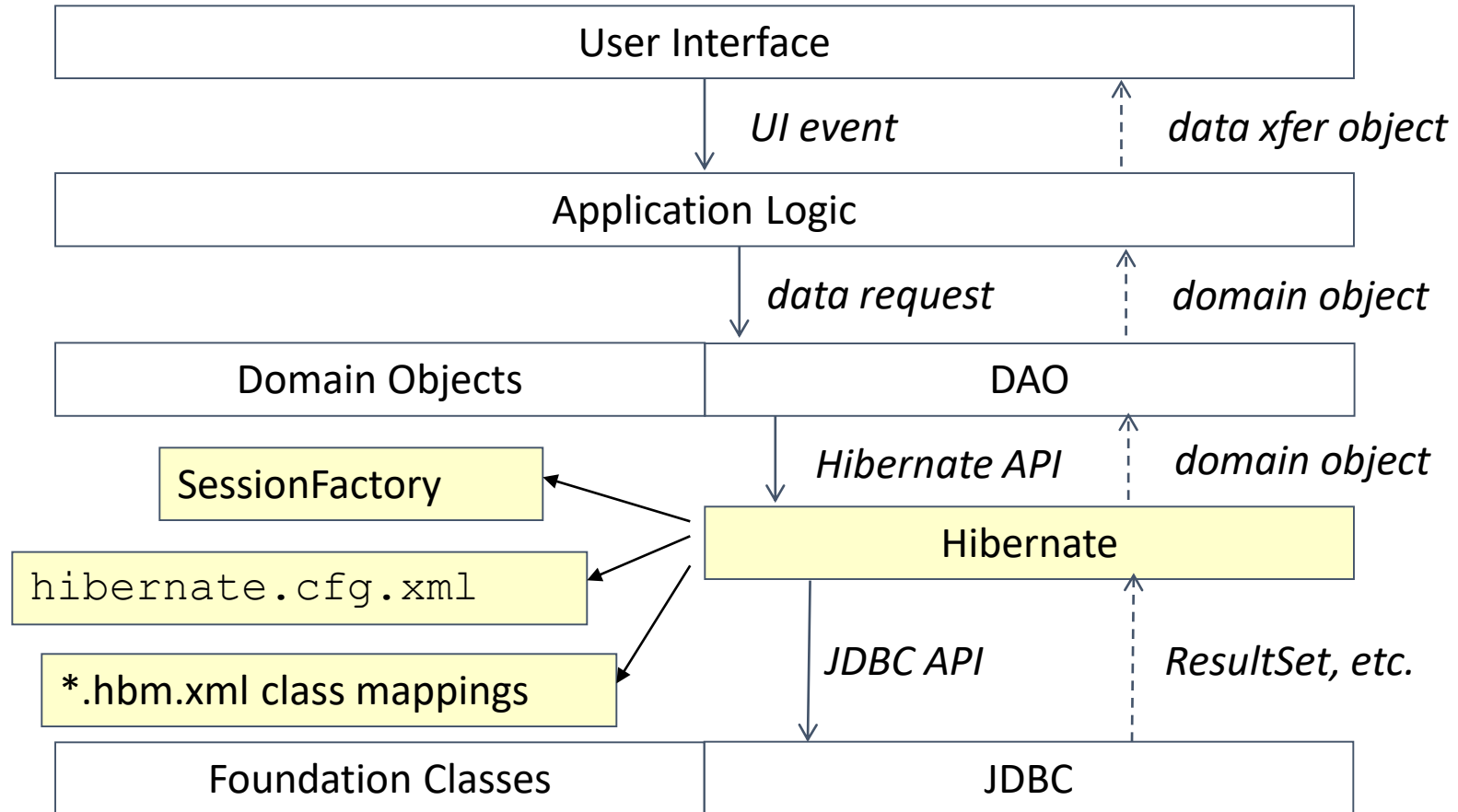
- DB2, FrontBase, HSQLDB, Informix, Ingres, Interbase, Mckoi, MySQL, Oracle, Pointbase, PostgreSQL, Progress, SAP DB, SQL Server, Sybase, etc...
- Or you can extend one of the `Dialect` classes

- Has a huge feature list
 - Go to their web site – we don't have time
- Maps JavaBeans (POJOs) to tables
 - XML defines the mappings
 - Very few bean requirements
- SQL is generated at app startup time
 - As opposed to bytecode manipulation
 - New database? No problem! Change a few props

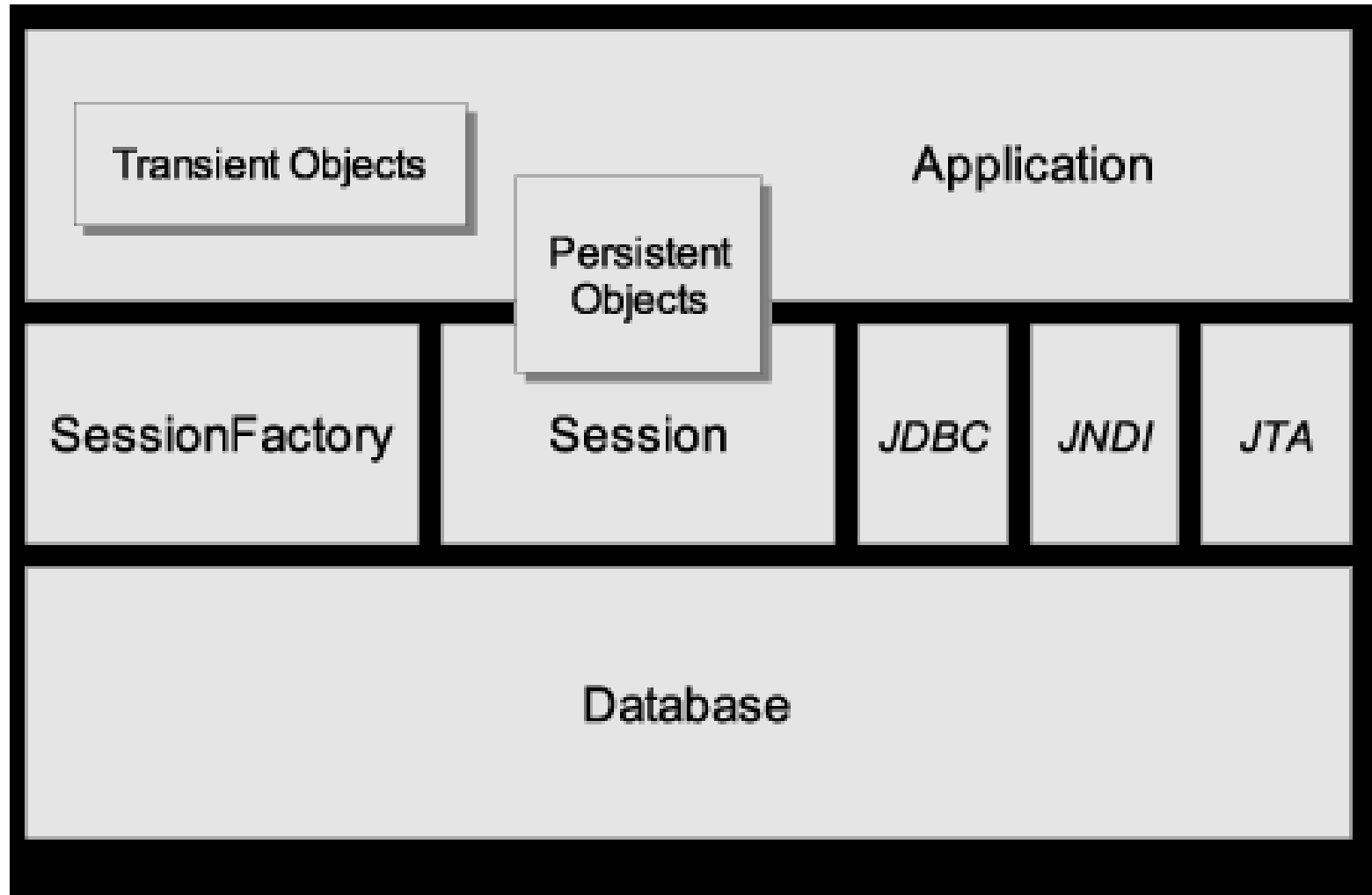
Hibernate Features

- O-R mapping using ordinary JavaBeans
- Can set attributes using *private* fields or *private* setter methods
- Lazy instantiation of collections (configurable)
- Polymorphic queries, object-oriented query language
- Cascading persist & retrieve for associations, including collections and many-to-many
- Transaction management with rollback
- Can integrate with other container-provided services

Application Architecture



Another View



Source: Hibernate Reference Manual (online)

Example of Persisting an Object

```
// get a Hibernate SessionFactory for Session management
SessionFactory sessionFactory = new Configuration()
    .configure().buildSessionFactory();

// an Event object that we want to save
Location tu = new Location( "Technical University" );
tu.setAddress( "Sofia" );
Event event = new Event("Java Days");
event.setLocation( tu );

Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
session.save( event );
tx.commit();
session.close();
```


Example of Retrieving an Object

```
// use the existing session factory
Session session = sessionFactory.openSession();

// query Event objects for "Java Days"
Transaction tx = session.beginTransaction();
Query query = session.createQuery(
    "from Event where name='Java Days'" );
List events = query.list();
out.println("Upcoming Java Days events: ");
for( Object obj : events ) {
    Event event = (Event) obj;
    String name = event.getName();
    Location loc = event.getLocation();
    ...
}
tx.commit();
```

Using Named Parameters in Query

```
// use the existing session factory
Session session = sessionFactory.openSession();

// Hibernate Query Language (HQL) can use named params
Query query = session.createQuery(
    "from Event where name=:name");
query.setParameter( "name", "Java Days");
List events = query.list( );

out.println("Upcoming Java Days events: ");
for( Object obj : events ) {
    Event event = (Event) obj;
    String name = event.getName( );
    Location loc = event.getLocation( );
    ...
}
```

Exercise 1

- Configure a project with Hibernate
 - create an EntityManager project
 - add Hibernate libraries to the project
 - add JAR for database driver
 - create a hibernate.cfg.xml file
 - create log4j.properties

Project Configuration for Hibernate

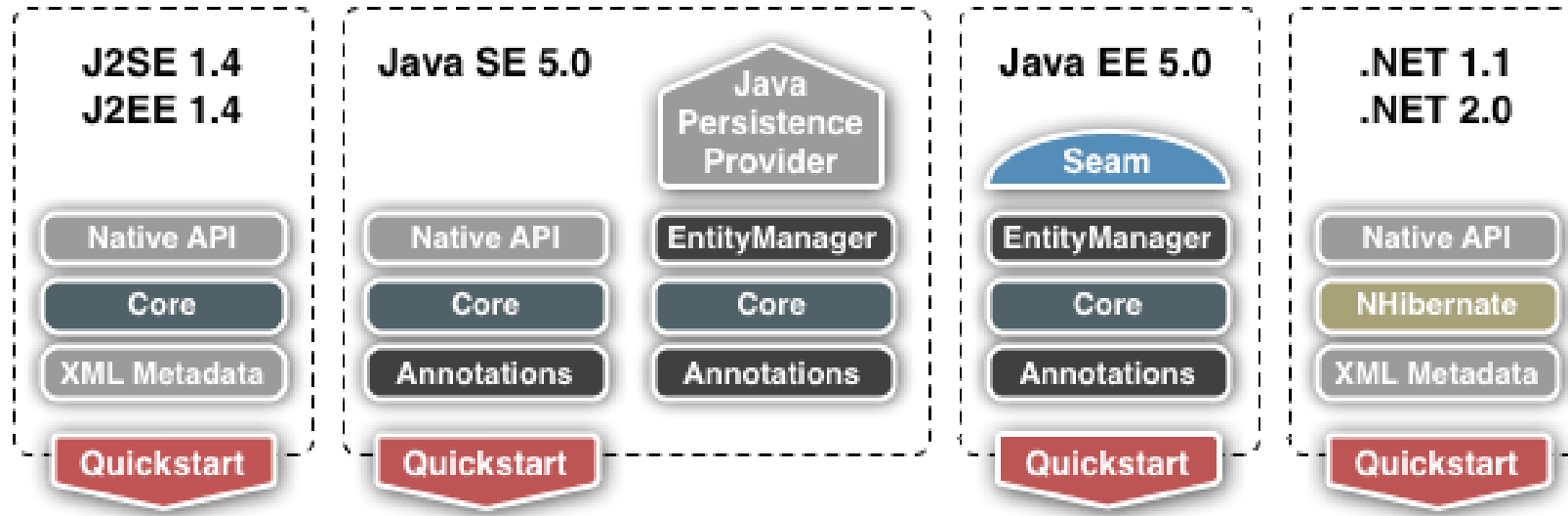
EventManager/	the project base directory
src/	
hibernate.cfg.xml	Hibernate configuration file
log4j.properties	Log4J configuration file
eventmgr/	base package is "eventmgr"
domain/	
Location.java	
location.hbm.xml	O-R mapping file for a class
build/	
hibernate.cfg.xml	copied here by IDE during build
log4j.properties	copied here by IDE during build
eventmgr/	
domain/	
Location.class	
location.hbm.xml	copied here by IDE during build
lib/	
hibernate3.jar	Hibernate requires several jars
asm.jar	
...	

Where to Get Hibernate



<http://www.hibernate.org>

Local copy:
<http://se.cpe.ku.ac.th/download/hibernate>



Required Hibernate Libraries

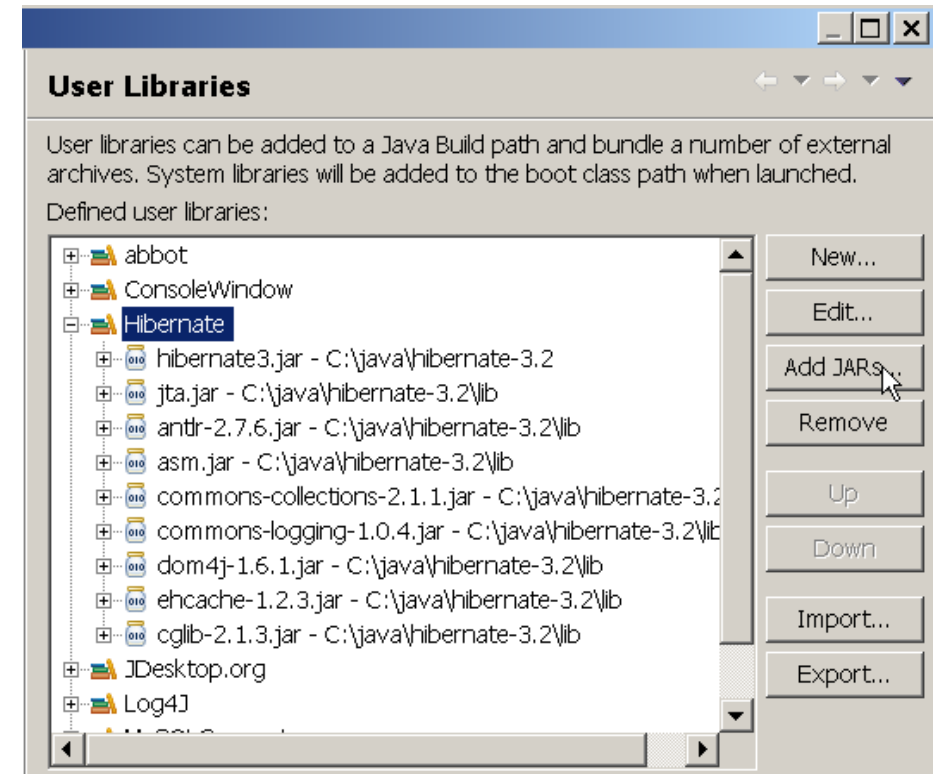
- Libraries are in **lib/** directory of Hibernate distribution.
- Which ones do you need?
 - See `_README.txt` or my *Using Hibernate* notes.
- In your IDE:
 - create a Hibernate "library"
 - add JARs to the library
 - better than copying JAR files to your project

Required Libraries

hibernate3.jar
antlr-2.7.6.jar
asm.jar
cglib.jar
commons-collections.jar
commons-logging.jar
dom4j.jar
ehcache.jar
jta.jar
log4j.jar
// and maybe
xml-apis.jar

Create a Hibernate Library:

1. Project -> Properties -> Java Build Path
2. select **Libraries** tab.
3. Click "Add Library..." and select "User Library", Next>
4. Create a new user library named "Hibernate"
5. Add Jar files



Add Library or jar for Database Driver

- For Embedded Derby Database

`/path-to-derby/lib/derby.jar`

- For HSQLDB

`/path-to-hsqldb/lib/hsqldb.jar`

hibernate.cfg.xml for Embedded Derby DB

```
<?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="dialect">
      org.hibernate.dialect.DerbyDialect</property>
    <property name="connection.driver_class">
      org.apache.derby.jdbc.EmbeddedDriver</property>
    <property name="connection.url">
      jdbc:derby:/temp/eventmgr;create=true</property>
    <property name="hbm2ddl.auto">create</property>
    <!-- O-R mappings for persistent classes -->
    <mapping resource="eventmgr/domain/Location.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```



Remove after database created

hibernate.cfg.xml for MySQL

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC ... remainder omitted >
<hibernate-configuration>
  <session-factory>
    <property name="dialect">
      org.hibernate.dialect.MySQLDialect </property>
    <property name="connection.driver_class">
      com.mysql.jdbc.Driver </property>
    <property name="connection.username">student</property>
    <property name="connection.password">secret </property>
    <property name="connection.url">
      jdbc:mysql://se.cpe.ku.ac.th/eventmgr </property>
    <!-- Object-Relational mappings for classes -->
    <mapping resource="eventmgr/domain/Location.hbm.xml"/>
    ... other mappings omitted
  </session-factory>
</hibernate-configuration>
```

XML Syntax, 1

XML is case sensitive

- XML Declaration: "this file contains XML"
`<?xml version='1.0' encoding='utf-8'?>`
- Root Element: "this whole file is a hibernate-configuration"
`<hibernate-configuration>`
content of document goes here
`</hibernate-configuration>`
- Child Elements: define content, have a tree-like nested structure
`<session-factory>`
...
`</session-factory>`
- Tree structure & Scope: all elements must have a closing tag
`<class name="Event" table="EVENTS">` *a start element*
`<property name="location" />` *element start & end*
`</class>` *an end tag*

XML Syntax, 2

- **Attributes:** values must always be quotes, no duplicates

```
<class name="Event" table="EVENTS">
```

- **Special characters:** < > & ' " must use reference except in CDATA

```
<message>
```

```
    &quot;Hello &lt;b>world&lt;/b>&quot;
```

```
</message>
```

- **Child Elements can be repeated (depends on DTD)**

```
<courselist name="courses">
```

```
    <course>object oriented programming</course>
```

```
    <course>software spec & design</course>
```

```
</courselist>
```

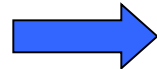
- **Elements can sometimes be written as attributes**

```
<course>
```

```
    <id>219341</id>
```

```
<course id="219341"/>
```

```
</course>
```



Logging

- Hibernate apps will log errors and/or activity.
- Two choices:
 - Java SDK logging (since JDK 1.4)
 - Log4j
 - if you use Log4j, create a `log4j.properties` in your application source root directory.
 - Copy this file from the Hibernate `etc/` directory.

Sample log4j.properties

- Too long to print here
- See an actual file, for example:

```
[hibernate.dir]/doc/tutorial/src/log4j.properties
```

- Configuration logging of Hibernate:

```
log4j.logger.org.hibernate=warn
```

- Log the SQL generated by Hibernate:

```
#log4j.logger.org.hibernate.SQL=debug
```

- Log JDBC bind parameters (can be security leak):

```
log4j.logger.org.hibernate.type=info
```

Exercise 2

- Define a Location class and LOCATIONS table
- Write the class
- Create a mapping file for Location: Location.hbm.xml

Create the Location class

Location class

Location
id: int name: String address: String

LOCATIONS table (Hibernate can auto-generate this)

LOCATIONS		
PK	id	INTEGER
	name	VARCHAR(80)
	address	VARCHAR(160)

Write the Location class

```
package eventmgr.domain;

public class Location {

    private int id;
    private String name;
    private String address;

    public Location() { } // a no argument constructor

    public int getId( ) { return id; }
    public void setId( int id ) { this.id = id; }
    public String getName( ) { return name; }
    public void setName( String n ) { this.name = n; }
    public String getAddress( ) { return address; }
    public void setAddress( String a ) { address = a; }

}
```

Use JavaBean conventions in Persistent object classes.

Hibernate can access private methods

```
package eventmgr.domain;

public class Location {

    private int id;
    private String name;
    private String address;

    public Location() { }

    public int getId( ) { return id; }
    private void setId( int id ) { this.id = id; }
    public String getName( ) { return name; }
    private void setName( String n ) { this.name = n; }
    public String getAddress( ) { return address; }
    private void setAddress( String a ) { address = a; }

}
```

OK to use "private" or
"protected" for *mutators*.

Hibernate can access private data, too

```
public class Location {  
    private int id;  
    private String name;  
  
    private void setName( String name ) {  
        if ( name.length() < 3 )  
            new RuntimeException("name too short");  
        ...  
    }  
}
```

Some mutator methods contain data validity checks or other complicated logic.

to tell Hibernate to set the field values directly (don't use the "set" method) in the class mapping file write:

```
<hibernate-mapping default-access="field">  
    ...  
</hibernate-mapping>
```

Schema to create Locations table

This works for MySQL.

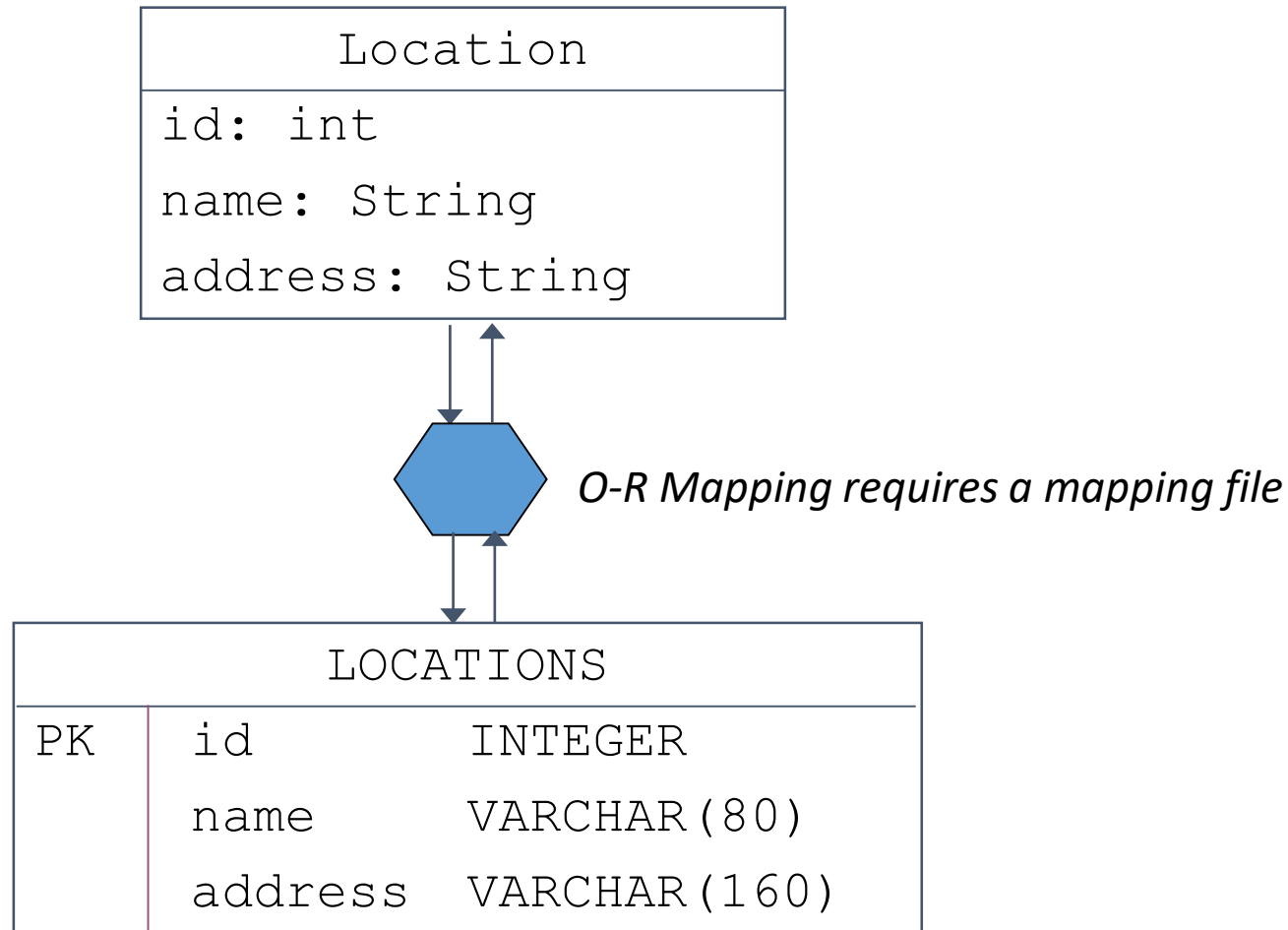
Hibernate can generate table schema at runtime.

```
CREATE TABLE locations (  
    id          INTEGER NOT NULL,  
    name        VARCHAR(80) NOT NULL,  
    address     VARCHAR(160),  
    PRIMARY KEY(id)  
) DEFAULT CHARSET=utf8 ;
```

```
mysql> use eventmgr ;  
mysql> source tableschema.sql ;
```

O-R Mapping for the Location class

Map between object attributes and table columns.



Mapping File Location.hbm.xml

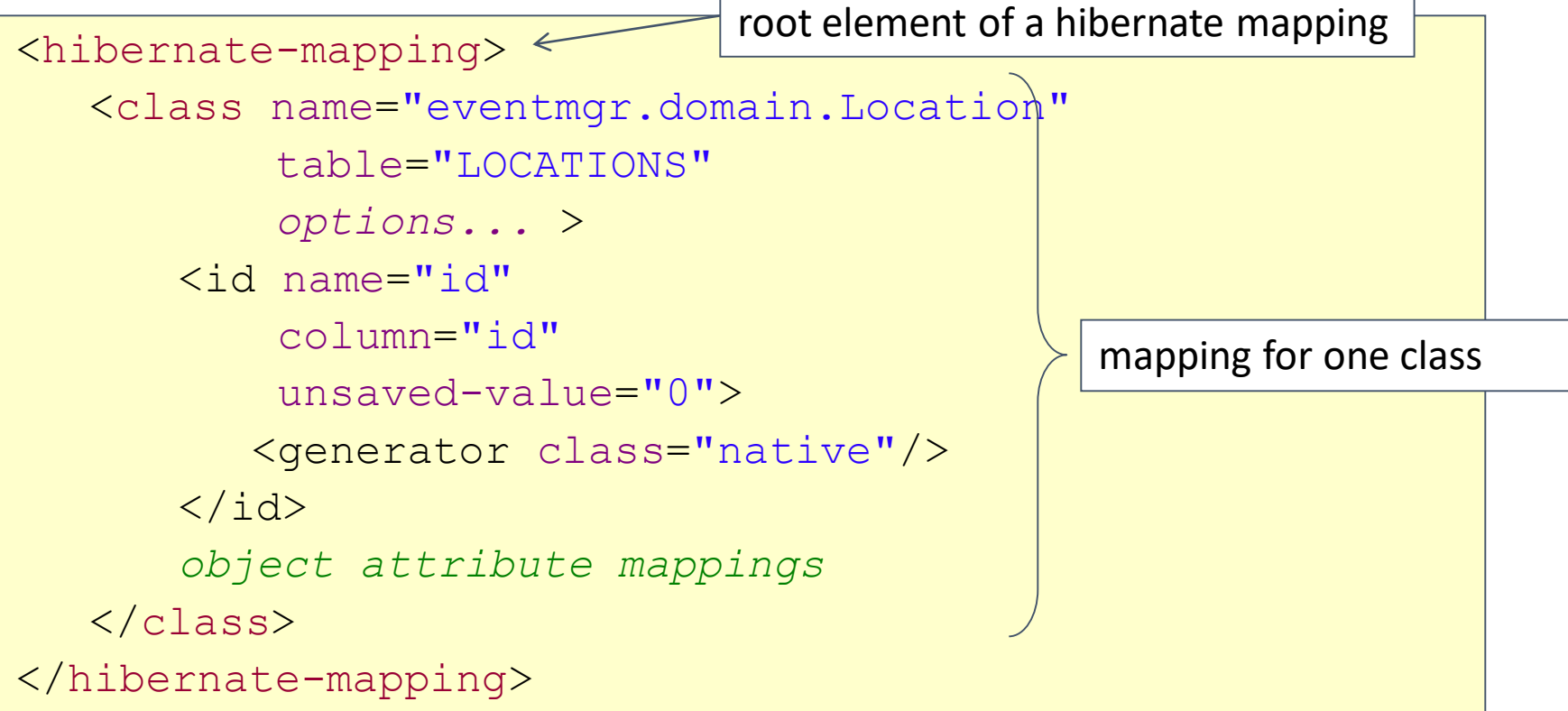
An XML file describing how to map object to table.

Filename: **Location.hbm.xml**

```
<!DOCTYPE hibernate-mapping PUBLIC ... remainder omitted >

<hibernate-mapping package="eventmgr.domain">
  <class name="Location" table="LOCATIONS">
    <id name="id" column="id">
      <!-- let hibernate choose id for new entities -->
      <generator class="native"/>
    </id>
    <property name="name" column="name" not-null="true"/>
    <property name="address"/>
  </class>
</hibernate-mapping>
```

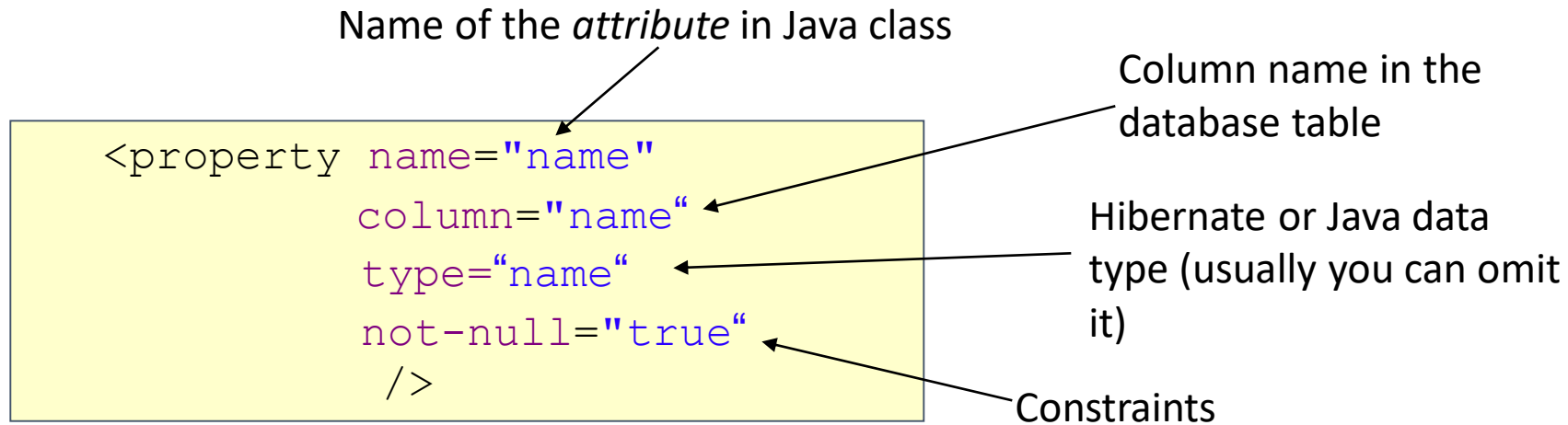
Mapping File Explained



Every persistent class needs an "identifier" attribute and column.

The identifier is used to establish object identity (`obj1 == obj2`) and locate the table row for a persisted object. The `id` is usually the Primary Key of the table.

Attribute Mapping: <property .../>



You omit elements if Hibernate can guess the value itself:

```
<property name="address" column="ADDRESS" type="string"/>
<!-- omit data type and Hibernate will determine it -->
<property name="address" column="ADDRESS"/>
<!-- omit column if same as attribute (ignoring case)-->
<property name="address"/>
```


What you have so far

- Project configured for Hibernate
- Hibernate configuration file
- Location class and mapping file Location.hbm.xml
- Configure a database schema
(for learning, we will auto-generate schema)

HibernateUtil: a Session Factory (1)

```
// always use the same sessionFactory (Singleton)
SessionFactory sessionFactory = new Configuration()
    .configure().buildSessionFactory();

// use sessionFactory to get a Session
Session session = sessionFactory.openSession();

// or the more convenient:
Session session = sessionFactory.getCurrentSession();
```

HibernateUtil	1
<u>-sessionFactory</u> : SessionFactory	
<u>+getSessionFactory</u> (): SessionFactory	
<u>+getCurrentSession</u> (): Session	
<u>+openSession</u> (): Session	

HibernateUtil: a Session Factory (2)

```
public class HibernateUtil {  
    private static SessionFactory sessionFactory = null;  
    private static final Logger log = Logger.getLogger(..);  
  
    public static SessionFactory getSessionFactory() {  
        if ( sessionFactory == null ) {  
            try {    // Create the SessionFactory  
                sessionFactory = new Configuration()  
                    .configure().buildSessionFactory();  
            } catch (Exception e) {  
                System.err.println("sessionFactory error "+ ex);  
                log.error("SessionFactory creation failed", ex);  
                //throw new ExceptionInInitializerError(ex);  
            }  
        }  
        return sessionFactory;  
    }  
}
```

HibernateUtil: a Session Factory (3)

```
/**
 * Get the current Hibernate session.
 * This creates a new session if no current session.
 * @return the current Hibernate Session
 */
public static Session getCurrentSession() {
    return getSessionFactory().getCurrentSession();
}

public static Session openSession() {
    return getSessionFactory().openSession();
}
```

Persistence Test: LocationTest.java

```
public class TestLocation {  
    public static void testSave() {  
        Location loc1 = new Location( );  
        loc1.setName("Kasetsart University");  
        loc1.setAddress("90 Pahonyotin Rd, Bangkok");  
        Location loc2 = new Location();  
        loc2.setName("UCLA");  
        loc2.setAddress("Westwood, California");  
  
        Session session =  
            HibernateUtil.getCurrentSession();  
        Transaction tx = session.beginTransaction();  
        session.saveOrUpdate( loc1 );  
        session.saveOrUpdate( loc2 );  
        tx.commit();  
        System.out.println("Locations saved");  
    }  
}
```

Persistence Test: LocationTest.java

```
public static void testQuery() {
    System.out.println("Retrieving locations");
    Session session = HibernateUtil.getCurrentSession();
    Transaction tx = session.beginTransaction();

    Query query = session.createQuery("from Location");
    // query.list() returns objects, cast to List<Location>
    List<Location> list = (List<Location>)query.list( );
    tx.commit();

    for(Location l : list ) out.printf("%d %s %s\n",
        l.getId(), l.getName(), l.getAddress() );
    if ( session.isOpen() ) session.close();
}

public static void main( String [] args) {
    testSave( ); testQuery( );
}
```

Exercise 3

- Test object uniqueness:
 - In one session get the same location two times and compare using (ku1 == ku2). Are they same?
 - In different sessions get the same location and compare (ku1 == ku2). Are they same?
- Test transparent persistence:
 - Modify a location inside a session. Does database change?
 - Reattach a modified object to new session. Does database change?

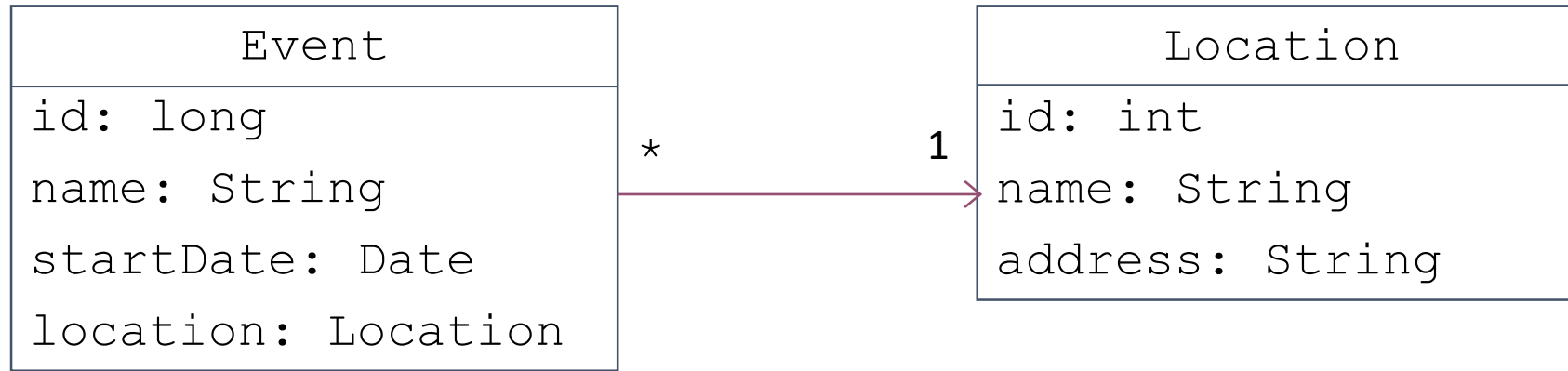
Getting a Unique Result

```
Session session = HibernateUtil.getCurrentSession();
Query query = session.createQuery(
    "from Location where name='Kasetsart University'");
// query.uniqueResult() returns only first match
Location kul = (Location) query.uniqueResult();
```

Pattern matches "like" - use "%" as wildcard character

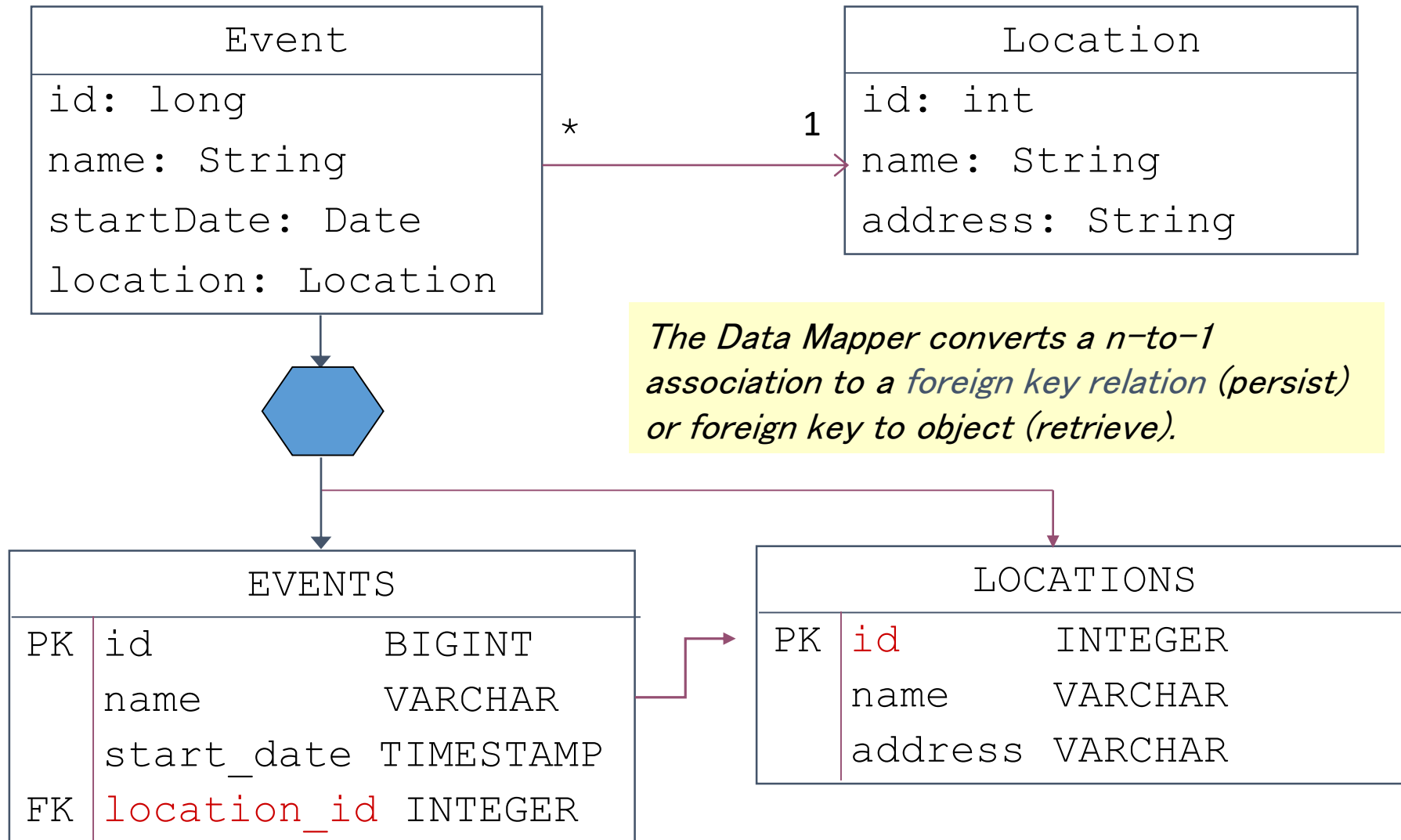
```
Session session = HibernateUtil.getCurrentSession();
Query query = session.createQuery(
    "from Location l where l.name like 'Kasetsart%'");
// query.uniqueResult() returns only first match
Location kul = (Location) query.uniqueResult();
```


Mapping a Class with Associations



Simplified version of Event class.

O-R Mapping of n-to-1 Associations

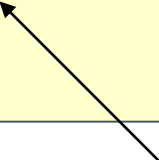


Mapping for Event (**Event.hbm.xml**)

Use `<many-to-one name="attribute" ... />`
to map a reference to another object.

```
<!DOCTYPE hibernate-mapping PUBLIC ... remainder omitted >
<hibernate-mapping package="eventmgr.domain">
  <class name="Event" table="EVENTS">
    ...

    <property name="startDate" column="start_date"
              type="timestamp" />
    <many-to-one name="location" column="location_id"
                 class="Location" />
  </class>
</hibernate-mapping>
```



you can omit **class** (Hibernate can determine itself)

Test: Save an Event

```
public static void testSave() {  
    Location loc1 = new Location( );  
    loc1.setName("Kasetsart University");  
    loc1.setAddress("90 Pahonyotin Rd, Bangkok");  
  
    Event event = new Event( );  
    event.setName("Java Days");  
    event.setLocation( loc1 );  
    event.setStartDate( new Date(108,Calendar.JULY, 1) );  
  
    Session session = HibernateUtil.getCurrentSession();  
    Transaction tx = session.beginTransaction();  
    session.saveOrUpdate( event );  
    tx.commit();  
    System.out.println("Event saved");  
}
```

Did you get an Error?

The Location doesn't exist in database (*transient object*).

```
Exception in thread "main"
```

```
    org.hibernate.TransientObjectException:
```

```
        object references an unsaved transient instance
```

```
- save the transient instance before flushing:
```

```
eventmgr.domain.Location
```

Persisting the Event location

Solutions:

1. save location during the transaction (manual save)
2. tell Hibernate to ***cascade*** the save operation (automatically save Location)

```
<many-to-one name="location" column="location_id"  
             class="Location"  
             cascade="save-update" />
```

cascade="none"

don't cascade operations

"all"

cascade all operations (be careful)

"save-update"

cascade save and updates

"delete-orphan"

cascade all, delete unreferenced orphan children


Test: Retrieve an Event

```
public static void testRetrieve() {  
    System.out.println("Retrieving event");  
    Session session = HibernateUtil.getCurrentSession();  
    Transaction tx = session.beginTransaction();  
  
    Query query = session.createQuery(  
        "from Event e where e.name= :name");  
    query.setParameter("name", "Java Days");  
    // query.list() returns objects, cast to List<Location>  
    List<Event> list = (List<Event>)query.list();  
    tx.commit();  
  
    for(Event e : list ) out.printf("%d %s %s\n",  
        e.getId(), e.getName(), e.getLocation().getName()  
    );  
}
```

Lazy Instances and Proxies

```
Transaction tx = session.beginTransaction();
Query query = session.createQuery(
    "from Event e where e.name=:name");
query.setParameter("name", "Java Days");
List<Event> list = (List<Event>)query.list( );
tx.commit();

for(Event e : list ) out.printf("%d %s %s\n",
    e.getId(), e.getName(), e.getLocation().getName()
);
```



Error: *LazyInstantiationException*

- Hibernate uses *lazy instantiation* and proxy objects
- Hibernate *instantiates* the location object when it is first accessed
- We closed the transaction *before* accessing the location

Two Solutions

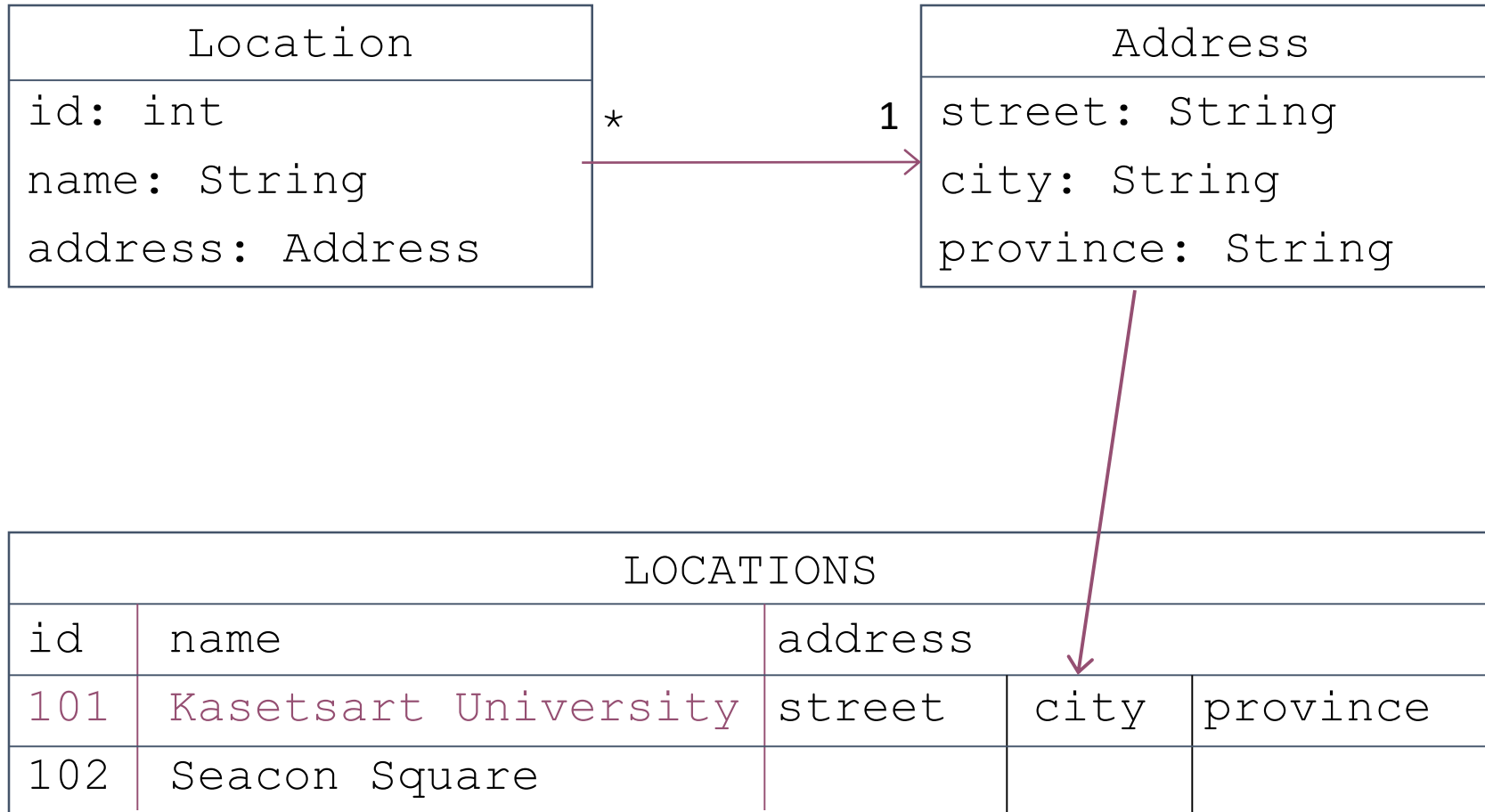
1. Modify our code: getLocation() before closing the session.

```
List<Event> list = (List<Event>)query.list( );  
for(Event e : list ) out.printf("%d %s %s\n",  
    e.getId(), e.getName(), e.getLocation().getName()  
);  
tx.commit( );
```

2. Tell Hibernate not to use lazy instantiation of Location objects (in Location.hbm.xml)

```
<class name="Location" table="LOCATIONS" lazy="false">  
    ...  
</class>
```

Creating a Component for Address



Organizing your Work

Use DAO to separate OR behavior from the rest of your project.

A DAO for the Location class

The "useful" methods depend on the domain class and the application.

```
LocationDao
```

```
findById(id: int) : Location
```

```
findByName(name : String): Location[*]
```

```
find(query: String) : Location[*]
```

```
save(loc: Location) : boolean
```

```
delete(loc: Location) : boolean
```

Java code for Simple LocationDao

```
package eventmgr.dao;  
import ...;  
public class SimpleLocationDao {  
    private static final Logger logger =  
        Logger.getLogger(LocationDao.class);  
  
    public LocationDao() { }  
    public Location findById( int id )  
    public List<Location> findByName( String name )  
    public List<Location> find( String query )  
    public boolean save( Location loc )  
    public boolean delete( Location loc )  
}
```

The core of findById() - use "load"

```
public Location findById( int id ) {  
    Location result = null;  
    Session session = null;  
    Transaction tx = null;  
    try {  
        session = HibernateUtil.getCurrentSession();  
        tx = session.beginTransaction();  
        result =  
            (Location) session.load(Location.class, id);  
        tx.commit();  
        session.close( );  
    } catch ...  
    }  
    return result;  
}
```

The details of findById()

```
public Location findById( int id ) {
    Location result = null;
    Session session = null;
    Transaction tx = null;
    try {
        session = HibernateUtil.getCurrentSession();
        tx = session.beginTransaction();
        result = (Location)session.load(Location.class,id);
        tx.commit();
    } catch (ObjectNotFoundException e) {
        logger.info("Object not found. id = "+id, e);
        if ( tx != null && ! tx.wasCommitted() ) tx.rollback();
    } catch (HibernateException e) {
        logger.error("Hibernate exception", e);
        if ( tx != null && ! tx.wasCommitted() ) tx.rollback();
    } finally {
        if ( session != null && session.isOpen() ) session.close();
    }
    return result;
}
```

The core of findByName() - "query"

```
public List<Location> findByName( String name ) {  
    List<Location> result;  
    ...  
    try {  
        session = HibernateUtil.getCurrentSession();  
        tx = session.beginTransaction();  
        Query query = session.createQuery(  
            "from Location where name=:name" );  
        query.setParameter( "name", name );  
        result = (List<Location>) query.list( );  
        tx.commit();  
        session.close( );  
    } catch ...  
    return result;  
}
```


Details of findByName()

- Exercise

The core of save() - "saveOrUpdate"

```
public boolean save( Location location ) {  
    boolean result = false;  
    ...  
    try {  
        session = HibernateUtil.getCurrentSession();  
        tx = session.beginTransaction();  
        session.saveOrUpdate( location );  
        tx.commit();  
        session.close( );  
        result = true;  
    } catch ...  
  
    return result;  
}
```

Details of save

- Exercise

The core of delete() - "delete"

```
public boolean delete( Location location ) {  
    boolean result = false;  
    ...  
    try {  
        session = HibernateUtil.getCurrentSession();  
        tx = session.beginTransaction();  
        session.delete( location );  
        tx.commit();  
        session.close( );  
        result = true;  
    } catch ...  
  
    return result;  
}
```

Redundant Code

- Every DAO method has the same boilerplate code:

```
Session session = null;
Transaction tx = null;
try {
    session = HibernateUtil.getCurrentSession();
    tx = session.beginTransaction();
    do the work here
    tx.commit();
} catch (ObjectNotFoundException e) {
    logger.info("Object not found. "+id, e);
    if ( tx != null && ! tx.wasCommitted() ) tx.rollback();
} catch (HibernateException e) {
    logger.error("Hibernate exception", e);
    if ( tx != null && ! tx.wasCommitted() ) tx.rollback();
} finally {
    if ( session != null && session.isOpen() ) session.close();
}
```

Factor out Redundant Code

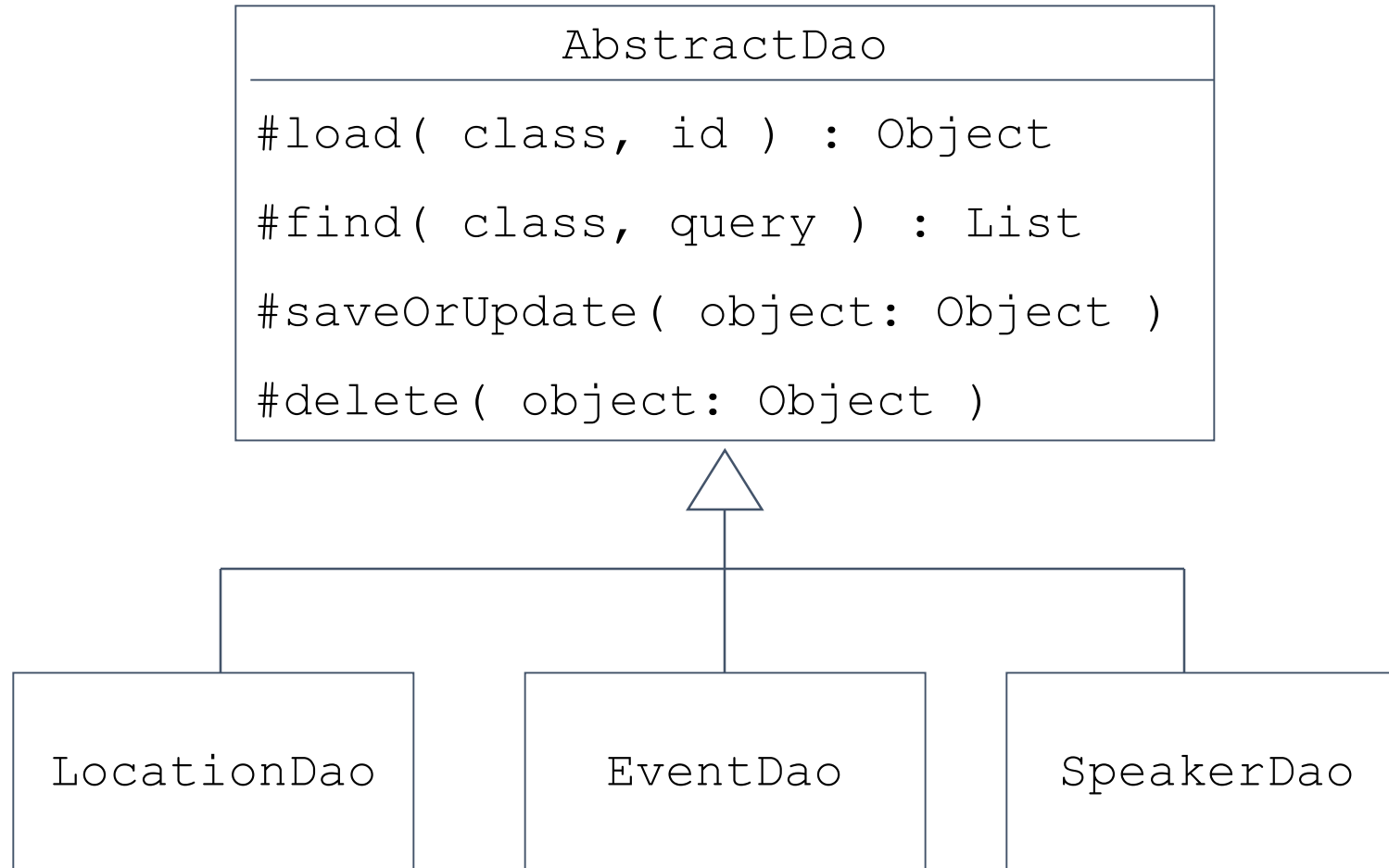
```
Class SimpleLocationDao {  
    private Session session; // declare as attributes  
    private Transaction tx;  
  
    public Location findById( int id ) {  
        try {  
            beginTransaction( );  
            do the work here  
            commitTransaction( );  
        } catch (ObjectNotFoundException e) {  
            handleError( e );  
        } catch (HibernateException e) {  
            handleError( e );  
        } finally {  
            if ( session != null && session.isOpen() )  
                session.close();  
        }  
    }  
}
```

Duplicate Code Between DAO

- In every DAO, the CRUD methods are almost the same
- Consider save() ...

```
public boolean save( Location location ) {  
    boolean result = false;  
  
    ...  
    try {  
        beginTransaction( );  
        session.saveOrUpdate( location );  
        commitTransaction( );  
    } catch ( ... ) {  
        handleException( e );  
    } finally { ... }  
    return result;  
}
```

Apply the Layer Superclass Pattern



AbstractDao.save

```
protected Object save( Object obj ) {  
    Object result = null;  
    try {  
        beginTransaction();  
        result = session.saveOrUpdate( obj );  
        commitTransaction();  
    } catch (ObjectNotFoundException e) {  
        handleError( e );  
    } catch (HibernateException e) {  
        handleError( e );  
    } finally {  
        closeSession( );  
    }  
    return result;  
}
```

LocationDao using Layer Superclass

```
public class LocationDao extends AbstractDao {  
  
    public boolean save( Location location ) {  
        return super.save( location );  
    }  
}
```

AbstractDao.load

```
protected Object load( Class clazz,
                      Serializable id ) {
    Object result = null;
    try {
        beginTransaction();
        result = session.load( clazz, id );
        commitTransaction();
    } catch (ObjectNotFoundException e) {
        handleError( e );
    } catch (HibernateException e) {
        handleError( e );
    } finally {
        closeSession( );
    }
    return result;
}
```

LocationDao using Layer Superclass

```
public class LocationDao extends AbstractDao {  
  
    public LocationDao( ) { super( ); }  
  
    public Location findById( int id ) {  
        return (Location) super.load(  
            Location.class, id );  
    }  
}
```

Exercise

- use your SimpleLocationDao to write a layer superclass named AbstractDao.
- write a LocationDao that extends AbstractDao

Hibernate Query Language (HQL)

```
Query query = session.createQuery(  
    "from Event where name='Java Days'");
```

```
Query query = session.createQuery(  
    "select e from Event e where e.name='Java Days'");
```

- Hibernate queries you Hibernate Query Language (HQL).
- HQL is object-centric - use class and property names, not SQL table names.

HQL example

Problem: Find all events which are held at Kasetsart

HQL:

```
String query = "select e from Event e where  
                e.location.name = 'Kasetsart University'";  
Query q = session.createQuery( query );
```

SQL and JDBC:

```
String sqlquery = "select * from EVENTS e  
                  join LOCATIONS l ON e.location_id = l.id  
                  where l.name = 'Kasetsart University'";  
Statement stmt = connection.createStatement( sqlquery );
```

Many-to-one Associations

```
public class Event {  
    private long id;  
    private String name;  
    private Location location; // 1-to-many assoc.  
    private Date startDate;  
    private long duration;  
  
    public Event( ) { }  
    ...  
}
```


Event.hbm.xml Mapping File

```
<!DOCTYPE hibernate-mapping PUBLIC ... remainder omitted >

<hibernate-mapping package="eventmgr.domain">
  <class name="Event" table="EVENTS">
    <id name="id" column="id">
      <generator class="native"/>
    </id>
    <property name="name" column="name" not-null="true"/>
    <property name="startDate" column="start_date"
      column="timestamp"/>
    <property name="duration" />
    <many-to-one name="location" column="location_id"
      class="Location"/>
  </class>
</hibernate-mapping>
```

Cascading Save & Update

When you save an Event, should Hibernate automatically save the location, too?

- no: then you must save the location yourself
- Yes: specify `cascade = "save-update"`

```
<class name="Event" table="EVENTS">
  <id name="id" column="id">
    <generator class="native"/>
  </id>
  ...
  <many-to-one name="location" column="location_id"
    class="Location"
    cascade="save-update"/>
</class>
```

Other choices for cascade: `all`, `none`, `save-update`, `delete`

Learning Hibernate

- Tutorial at www.hibernate.org
- Another good tutorial:
http://www.allapplabs.com/hibernate/hibernate_tutorials.htm
- Peak & Heudecker, *Hibernate Quickly*, 2006.
- Baur & King, *Java Persistence with Hibernate*, 2007, update of *Hibernate in Action*, and much longer.
- Baur & King, *Hibernate in Action*, 2005, considered the best Hibernate book, by one of the Hibernate creators.

Hibernate Tools

Hibernate Tools Eclipse plugin - HibernateConsole and more

- Test HQL queries, browse database

Middlegen - generates class mapping files (hbm.xml) from an existing database schema. Has Eclipse plugin.

Hibernate Synchronizer - generates class mapping files (hbm.xml) and Java classes from a database schema.

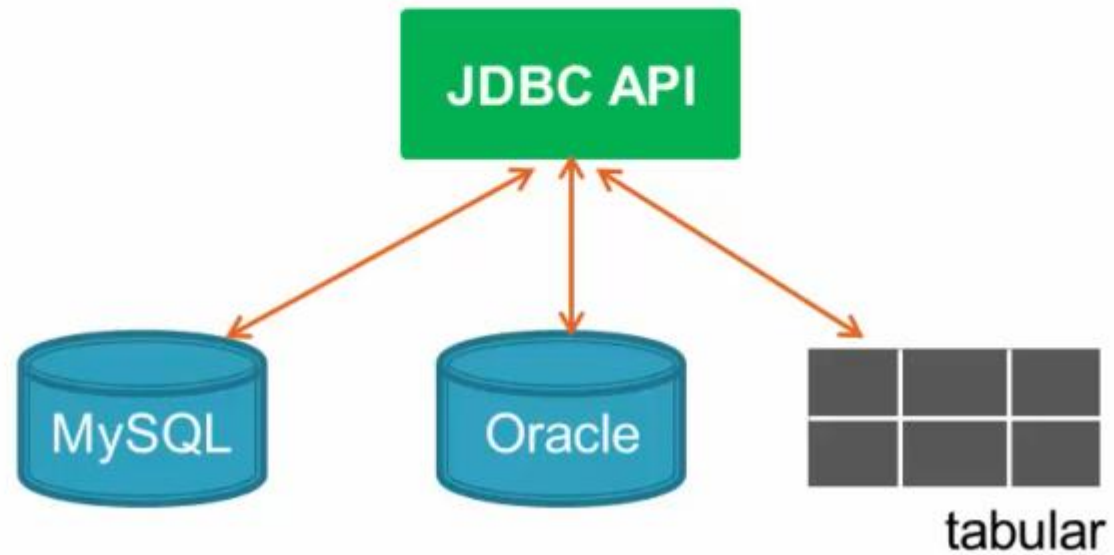
hbm2ddl - creates a database schema (in SQL) from a hbm.xml file.

hbm2java - creates a Java class from a hbm.xml file.

JDBC API & JDBC Driver

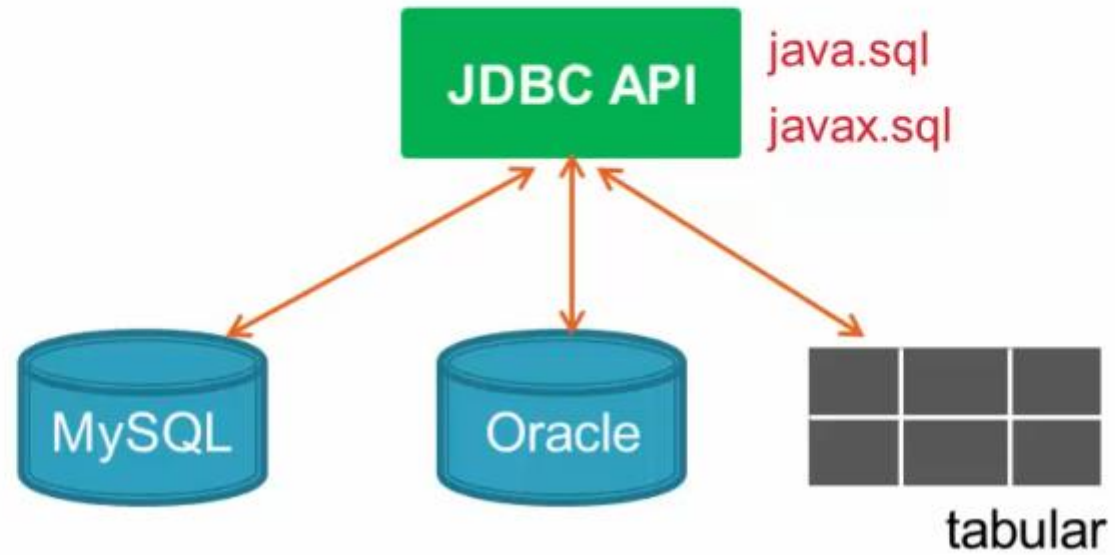
JDBC

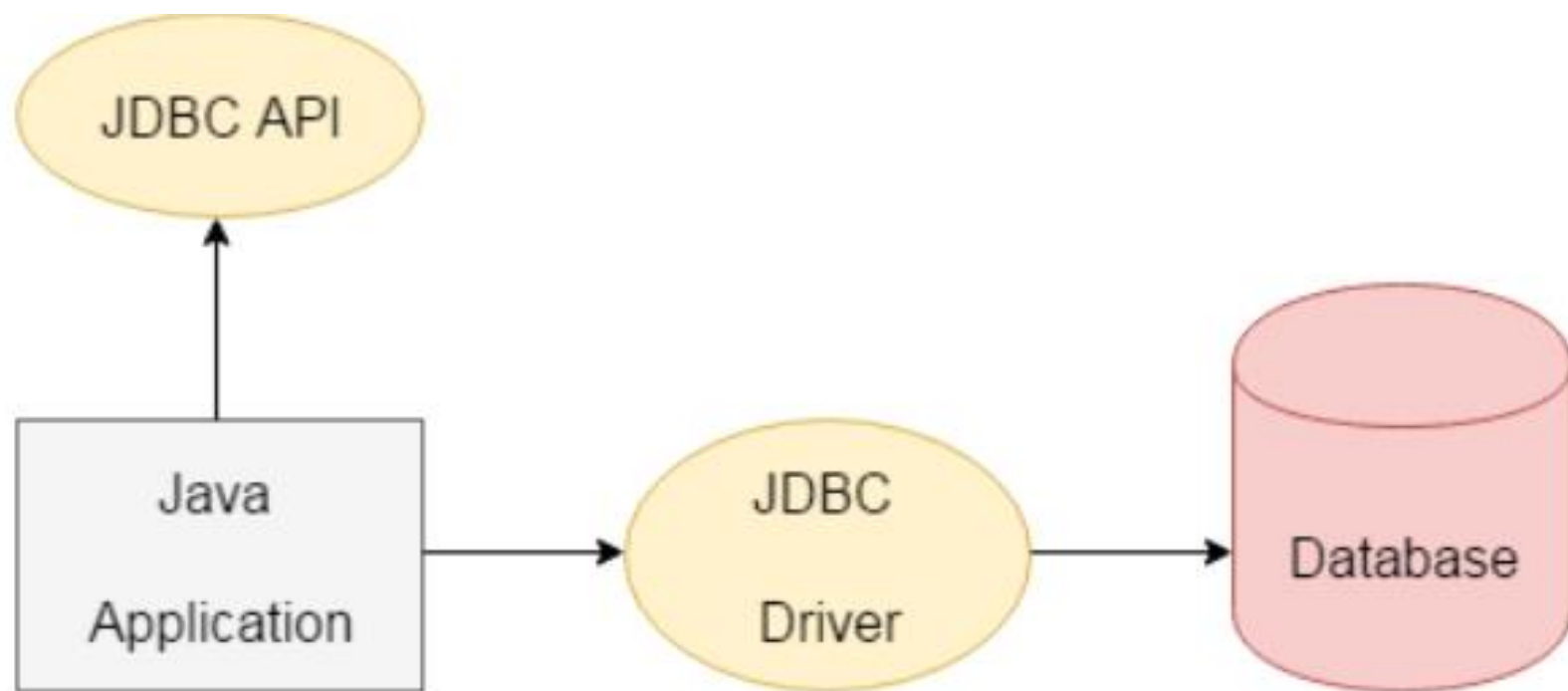
standard database-independent interface



JDBC

standard database-independent interface





The popular *interfaces* of JDBC API:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

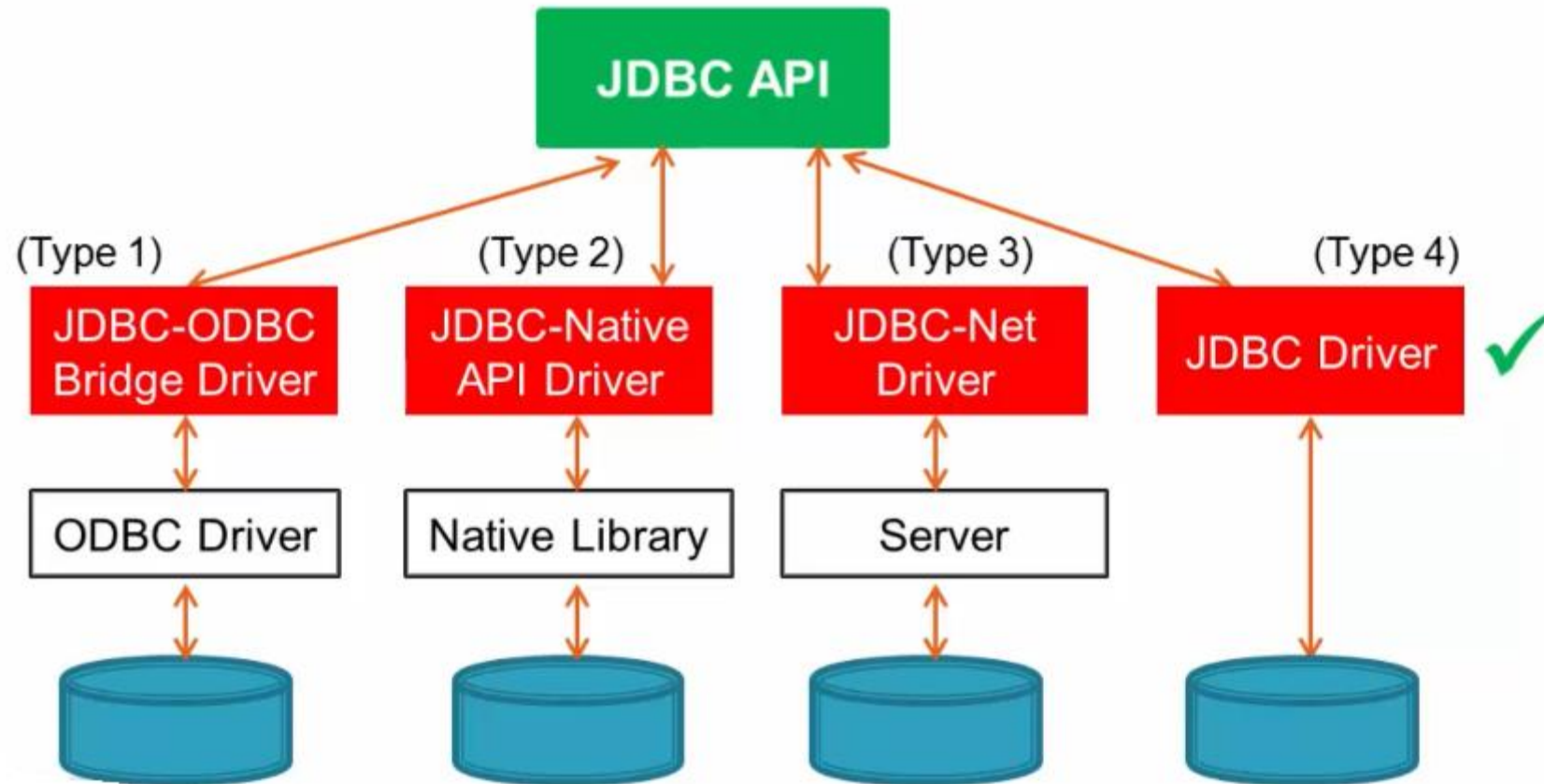
The popular classes of jdbc api:

- DriverManager class
- Blob class
- Clob class
- Types class

We can use JDBC API to handle database using Java program and can perform the following activities:

- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

JDBC Driver Types



Download MySQL Community Server

MySQL Community Edition is a freely downloadable version of the world's most popular open source database that is supported by an active community of open source developers and enthusiasts.

[MySQL Cluster Community Edition](#) is available as a separate download. The reason for this change is so that MySQL Cluster can provide more frequent updates and support using the latest sources of MySQL Cluster Carrier Grade Edition.

MySQL open source software is provided under the [GPL License](#).

OEMs, ISVs and VARs can purchase commercial licenses.



Important Platform Support Updates

Online Documentation:

- [Installation Instructions, Documentation and Change History](#) for the MySQL 5.7 Generally Available (GA) Release
- [Installation Instructions, Documentation and Change History](#) for the MySQL 5.6 Generally Available (GA) Release
- [Installation Instructions, Documentation and Change History](#) for the MySQL 5.5 Generally Available (GA) Release

Looking for previous GA versions?

- [MySQL Community Server 5.6 »](#)
- [MySQL Community Server 5.5 »](#)
- [Archived versions »](#)

MySQL Connectors

MySQL offers standard database driver connectivity for using MySQL with applications and tools that are compatible with industry standards ODBC and JDBC. Any system that works with ODBC or JDBC can use MySQL.

Connector/ODBC

Standardized database driver for Windows, Linux, Mac OS X, and Unix platforms.

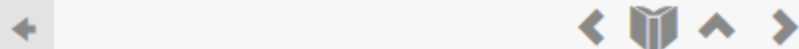
Connector/Net

Standardized database driver for .NET platforms and development.

Connector/J

MySQL open source software is provided under the [GPL License](#).

OEMs, ISVs and VARs can purchase commercial licenses.



Documentation Home

MySQL Connector/J 8.0 Developer Guide

- Preface and Legal Notices
- Overview of MySQL Connector/J
- Connector/J Versions, and the MySQL and Java Versions They Support
- What's New in Connector/J 8.0?

▼ Connector/J Installation

- Installing Connector/J from a Binary Distribution
- Installing Connector/J Using Maven
- Installing from Source
- Upgrading from an Older Version
- Testing Connector/J

MySQL Connector/J 8.0 Developer Guide / Connector/J Installation

Chapter 4 Connector/J Installation

Table of Contents

- 4.1 Installing Connector/J from a Binary Distribution
- 4.2 Installing Connector/J Using Maven
- 4.3 Installing from Source
- 4.4 Upgrading from an Older Version
- 4.5 Testing Connector/J

You can install the Connector/J package using either a binary or source distribution. While the binary distribution is the easiest to install, the source distribution lets you customize your installation. Both types of distributions are available. Connector/J is also available on GitHub at <https://github.com/mysql/mysql-connector-j>.

Connector/J is also available as a Maven artifact in the Central Repository. See [Section 4.2, “Installing Connector/J Using Maven”](#).

If you are upgrading from a previous version, read the upgrade information in [Section 4.4, “Upgrading from an Older Version”](#).

IDE interface showing a Java project named "thrillio" with a package structure including "src", "test", "JRE System Library [JavaSE-1.7]", "JUnit 4", "Referenced Libraries", "commons-lang3-3.4.jar", "pages", "Book", "Movie", "User", and "WebLink".

The main editor displays the file `DataStore.java` with the following code:

```
43
44     try {
45         Class.forName("com.mysql.jdbc.Driver");
46         new com.mysql.jdbc.Driver();
47
48         //System.out.println("Driver loaded");
49
50         //Driver driver = new Driver();
51         //driver = new Driver();
52     } catch (ClassNotFoundException e) {
53         e.printStackTrace();
54     }
55
56     // try-with-resources ==> conn & stmt will be closed
57     // Connection string: <protocol>:<sub-protocol>:<data-source details>
```

A tooltip is visible over line 46, indicating the error: "com.mysql cannot be resolved to a type". It suggests two quick fixes:

- Create class 'Driver' in package 'com.mysql.jdbc'
- Fix project setup...

The bottom panel shows the "Problems" view, indicating 2 references in the workspace (no JRE) (0 matches filtered from view) for the error.

com.semanticsquare.thrillio.bgjobs - src - thrillio

- WebpageDownloaderTask
- getWebLinks()

com.semanticsquare.thrillio.dao - src - thrillio

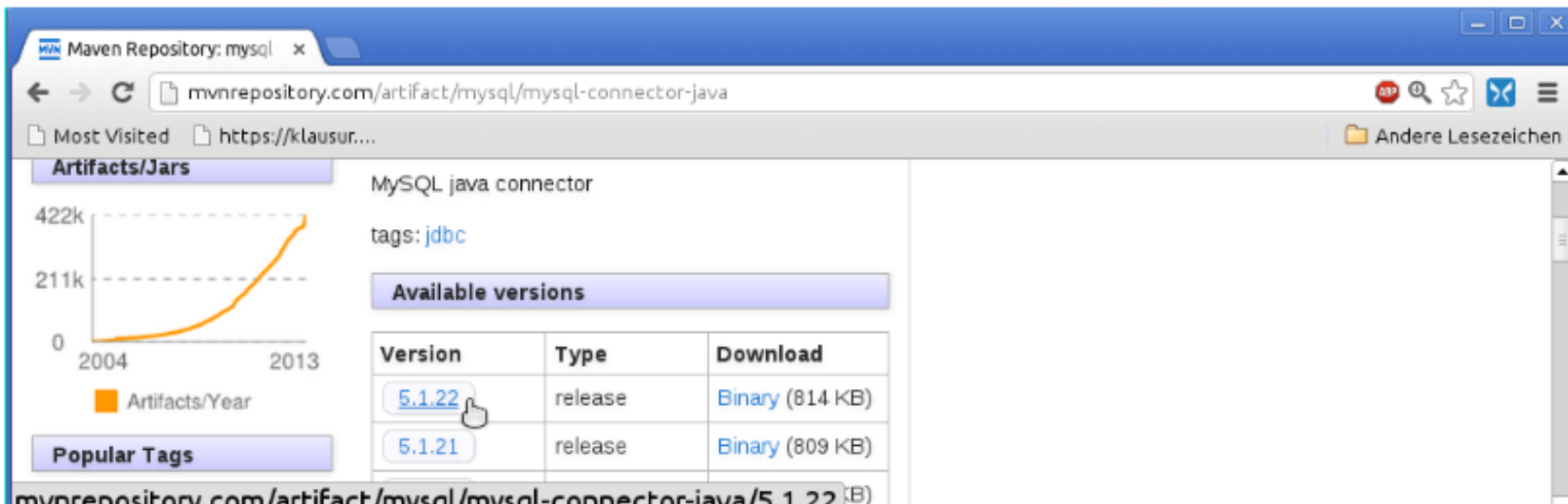
ИНСТАЛИРАНЕ НА Mysql™ JDBC™ драйвер

Трябва да свалим JDBC jar файла, подобно на `mysql-connector-java5.1.16.jar` и ръчно да го добавите в INTELLIJ.

Когато споделите проекта с други колеги и се ползва различен компютър, трябва този файл да е свален и на него. За целта има два варианта – или да се интегрира jar файла в проекта (lib директорията) и да се качи в Git, Svn. Другия вариант е в проекта да се добавят допълнителни (library) dependencies.

- https://freedocs.mi.hdm-stuttgart.de/sect_mavenAddMysqlJdbcConnector.html

добавяне допълнителни (library) dependencies



Maven Repository: mysql x

mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.22

Most Visited https://klausur....

home » mysql » mysql-connector-java » 5.1.22

Repository

- Plugins
- Tag Cloud

Artifacts/Jars

422k

211k

0

2004

2013

Artifacts/Year

Popular Tags

ajax analysis

annotations ant apache

api archetype aspect

asynchronously beans binding

bpm build buildsystem

bytecode cache cms

codecoverage codehaus

collections concurrency

MySQL java connector

MySQL java connector

Artifact	Download (JAR) (814 KB)
POM File	View
HomePage	http://dev.mysql.com/usingmysql/java/
Organization	
Issue Tracker	

Maven Ivy Grape Gradle Buildr SBT

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.22</version>
</dependency>
```

This artifact depends on ...

Undo

Redo

Cut

Copy

Paste

