Hibernate: Object – Relational Mapping

M. En C. José Asunción Enríquez Zárate
asuncionez@gmail.com
http://asuncionez.blogspot.com

ORACLE MySQL HIBERNATE JEDI

Contents

- ✓ The Problem
- ✓ Object-Relational Mapping
- ✓ Hibernate Core Concepts
- ✓ Hibernate Architecture
- ✓ Hibernate Mappings
- ✓ Configuring Hibernate
- ✓ Working with Persistent Objects
- ✓ Persistence Lifecycle
- ✓ Working with Collections
- ✓ HQL – The Hibernate Query Language
- ✓ Other Hibernate Features
- ✓ Conclusions

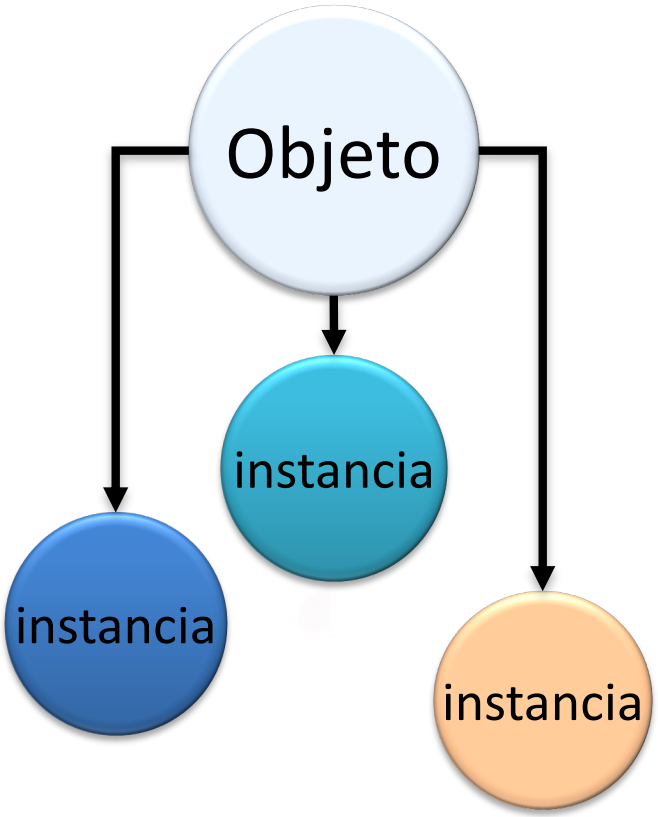
The Problem
Two Different Worlds

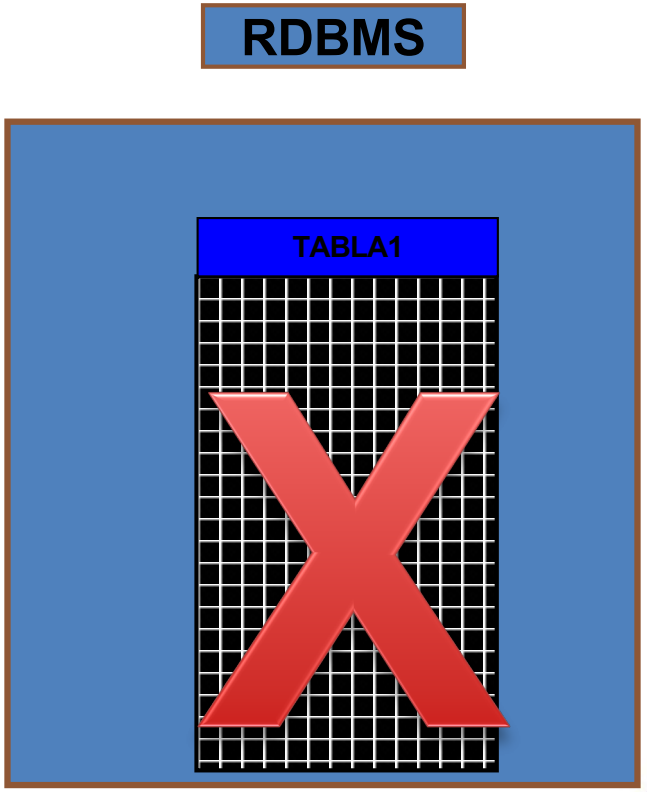
Object-Oriented Systems

- ✓ Represent a problem in terms of objects
- ✓ Semantically richer, encapsulate data and behavior

- ✓ **Relational Databases**
- ✓ Efficient data storage, retrieval and integrity of the data
- ✓ Provide a "normalized" set of data

The Problem
Two Different Worlds





ENTORNO DE OBJETOS

ENTORNO DE DATOS

The Problem
Object-Relational Mismatch

- ✓ Granularity
  - ✓ Object Oriented vs. Database Types
- ✓ Inheritance & Polymorphism
  - ✓ Not supported in the relational model
  - ✓ No polymorphic queries
  - ✓ Columns are strictly typed

The Problem

Object-Relational Mismatch

- ✓ Identity Mismatch
  - ✓ Different objects can map to the same column or columns
  - ✓ Database Identity Strategies
  - ✓ Natural Keys versus Surrogate Keys

The Problem

Object-Relational Mismatch

- ✓ Associations
  - ✓ Object References versus Foreign Keys
  - ✓ Directionality of Associations
  - ✓ Tables associations are always one-to-many or many-to-one
  - ✓ Object associations can be many-to-many

The Problem

Object-Relational Mismatch

- ✓ Object Graph Navigation
  - ✓ Object-oriented applications "walk" the object graph
  - ✓ Object-oriented systems access what they need
  - ✓ With database we need a "plan" of what to retrieve

Object-Relational Mapping

- ✓ Tools/techniques to store and retrieve objects from a database
- ✓ From the code perspective it behaves like a virtual object database
- ✓ A complete ORM solution should provide:
  - ✓ Basic CRUD functionality
  - ✓ An Object-Oriented Query Facility
  - ✓ Mapping Metadata support
  - ✓ Transactional Capability

Hibernate Relational Persistence For Idiomatic Java

- ✓ Created by Gavin King, Distinguished Engineer at IBM, RedHat & JBoss

Hibernate Relational Persistence For Idiomatic Java

- ✓ Provides
  - ✓ Transparent persistence
  - ✓ Object-querying capabilities
  - ✓ Caching
- ✓ Works with fine-grained POJO-based models
- ✓ Supports most Databases

Hibernate

Relational Persistence For Idiomatic Java

- ✓Declarative programming
- ✓Uses Runtime Reflection
- ✓Query Language is SQL-like
  - ✓Leverages programmer's knowledge
- ✓Mappings
  - ✓Typically defined as XML documents
- ✓But you never have to write XML if you don't want to.

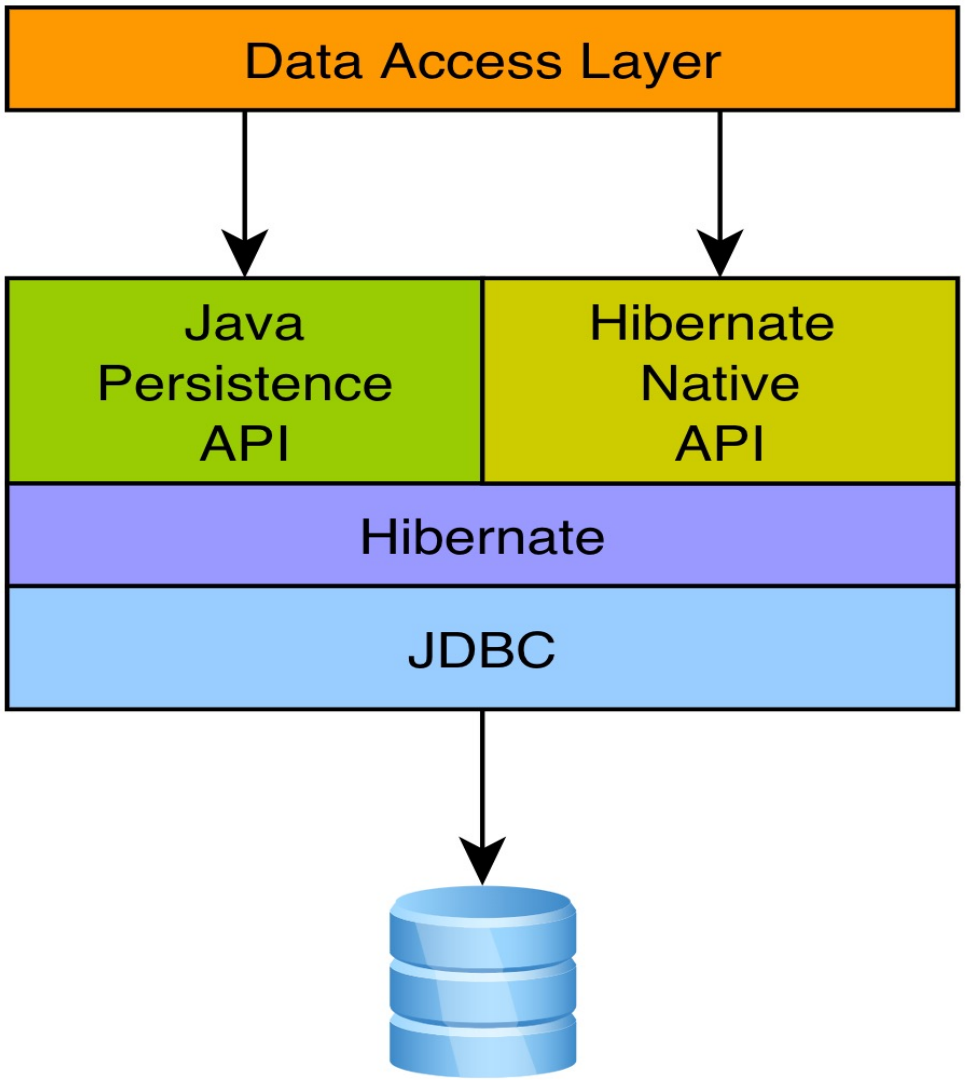
Hibernate

Relational Persistence For Idiomatic Java

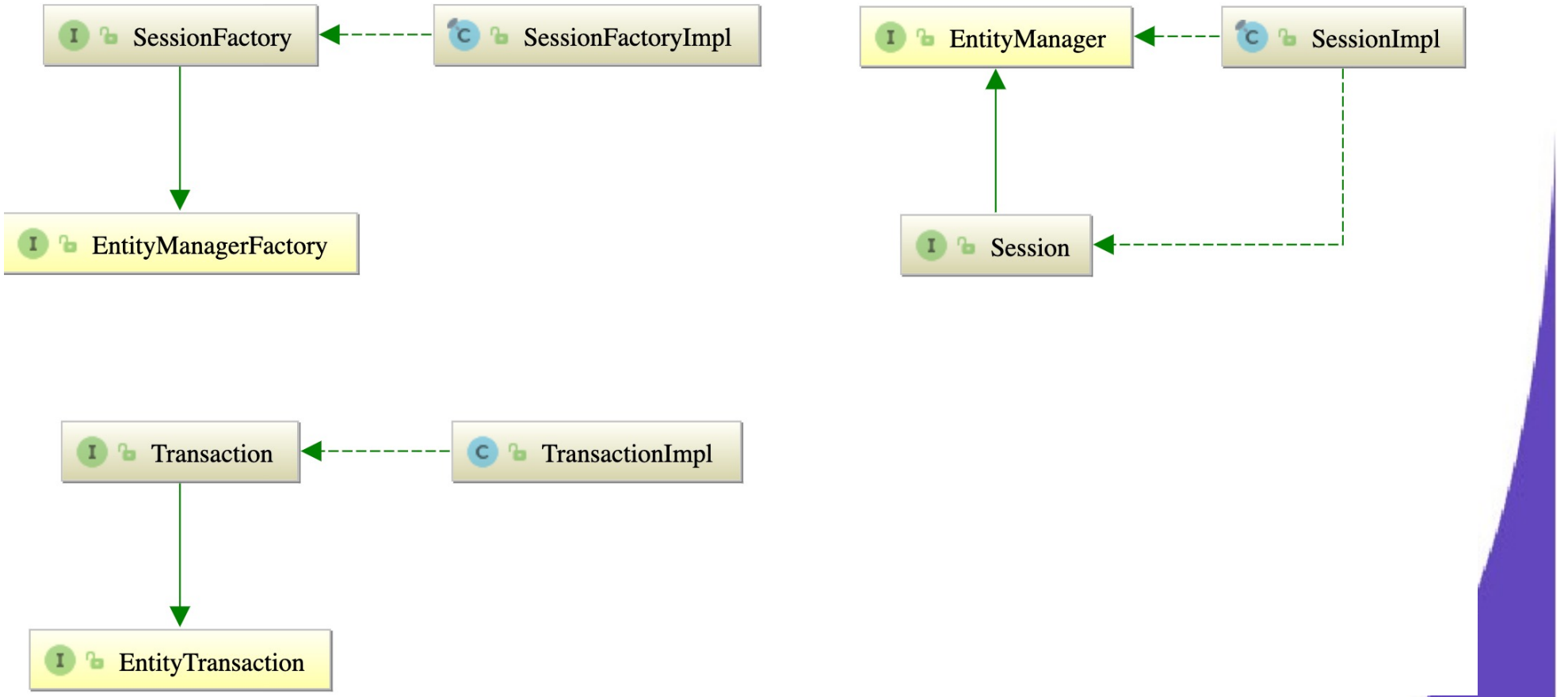
Hibernate
Core Concepts

- ✓ **Session Factory**
  - ✓ is a cache of compiled mappings for a single database.
  - ✓ used to retrieve Hibernate Sessions
- ✓ **Session**
  - ✓ Short lived
    - ✓ Temporary bridge between the app and the data storage
  - ✓ Provides CRUD operations on Objects
  - ✓ Wraps a JDBC Connection / JEE Data Source
  - ✓ Serves as a first level object cache

Architecture



Hibernate JPA Provider



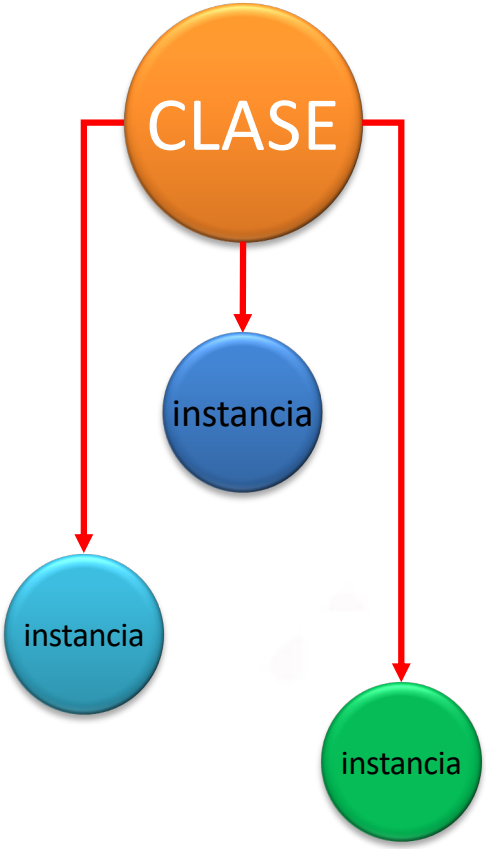
Architecture
Choices

- ✓ How much you use depends on your needs
  - ✓ Lightweight
    - ✓ Basic persistence
  - ✓ Fully Integrated
    - ✓ Transactions
    - ✓ Caching
    - ✓ Connection Pooling

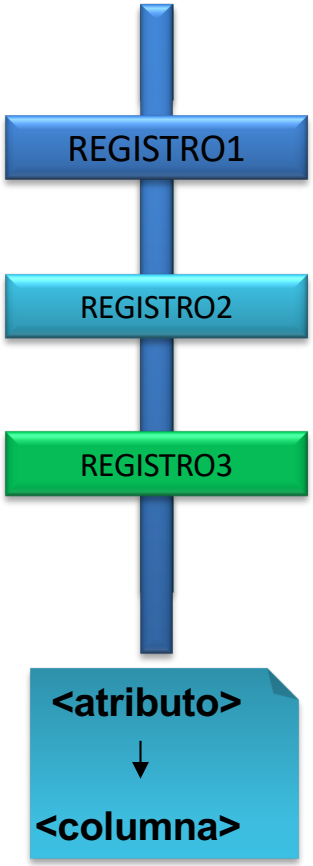
O/R Mappings

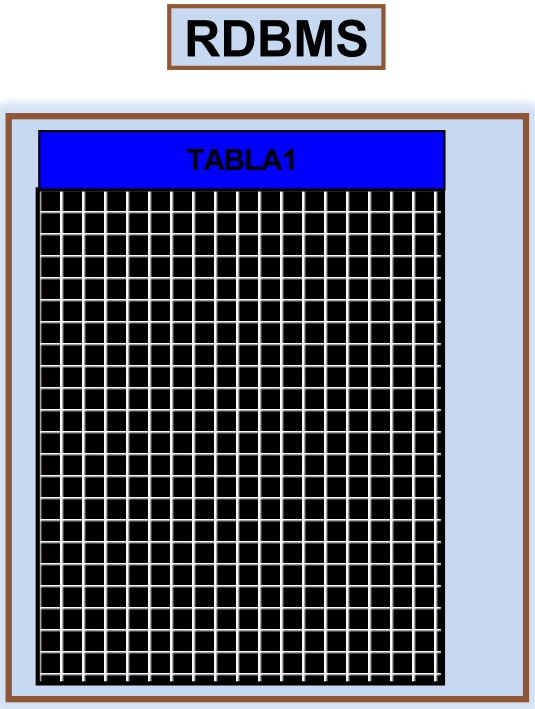
- ✓ Hibernate Mapping (.hbm.xml) Files define:
  - ✓ Column to Field Mappings
  - ✓ Primary Key mapping & generation Scheme
  - ✓ Associations
  - ✓ Collections
  - ✓ Caching Settings
  - ✓ Custom SQL
  - ✓ And many more settings…

Hibernate
Relational Persistence For Idiomatic Java



MAPEO O-R





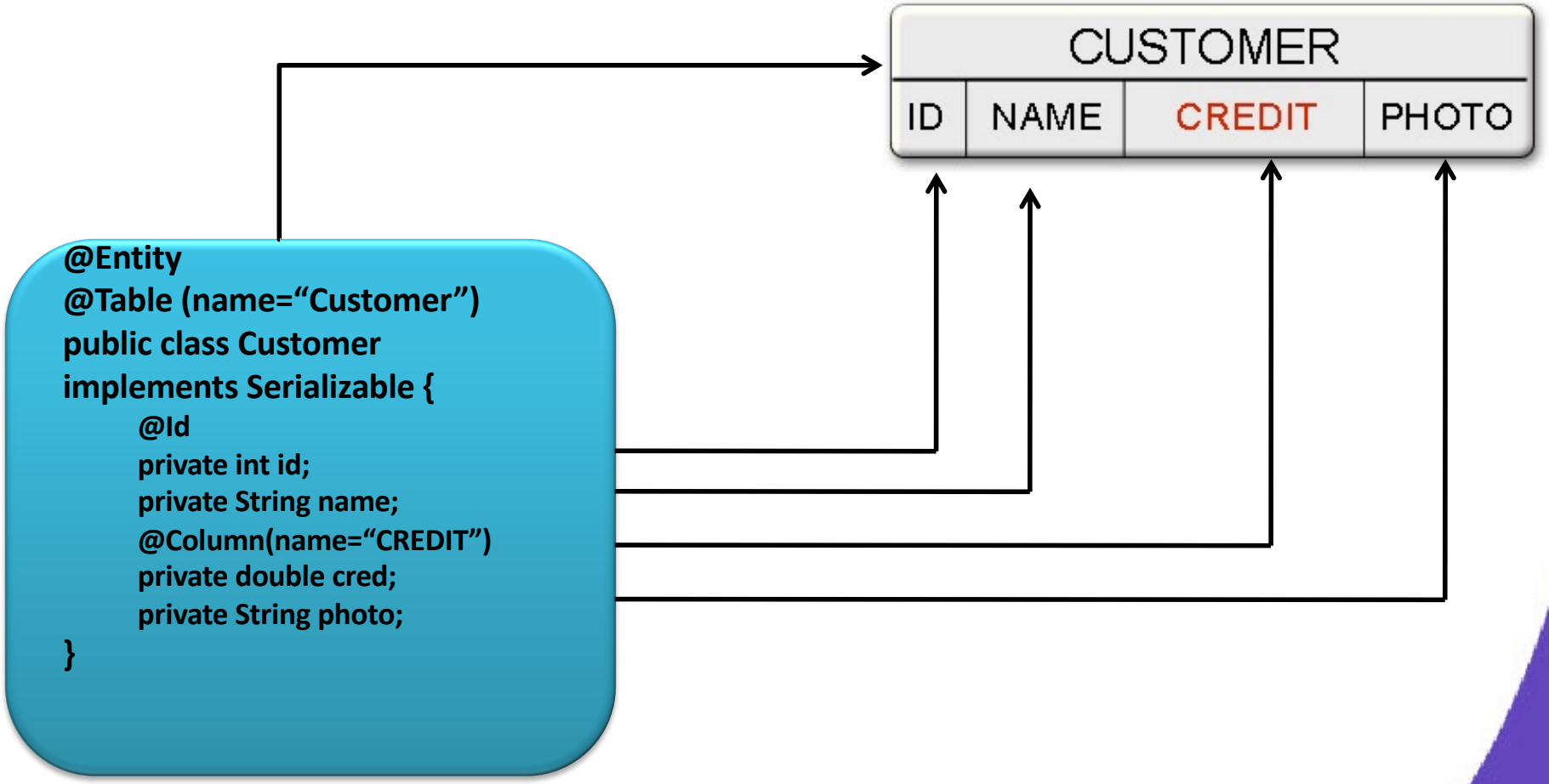
ENTORNO DE OBJETOS

ENTORNO DE DATOS

O/R Mappings

- ✓ Mapping files can be generated:
  - ✓ Manually
    - ✓ Tedious, boring and error prone
    - ✓ Total control of the mapping
  - ✓ From POJOs
    - ✓ XDoclet, Annotations
  - ✓ From Database Schema
    - ✓ Middlegen, Synchronizer, Hibernate Tools

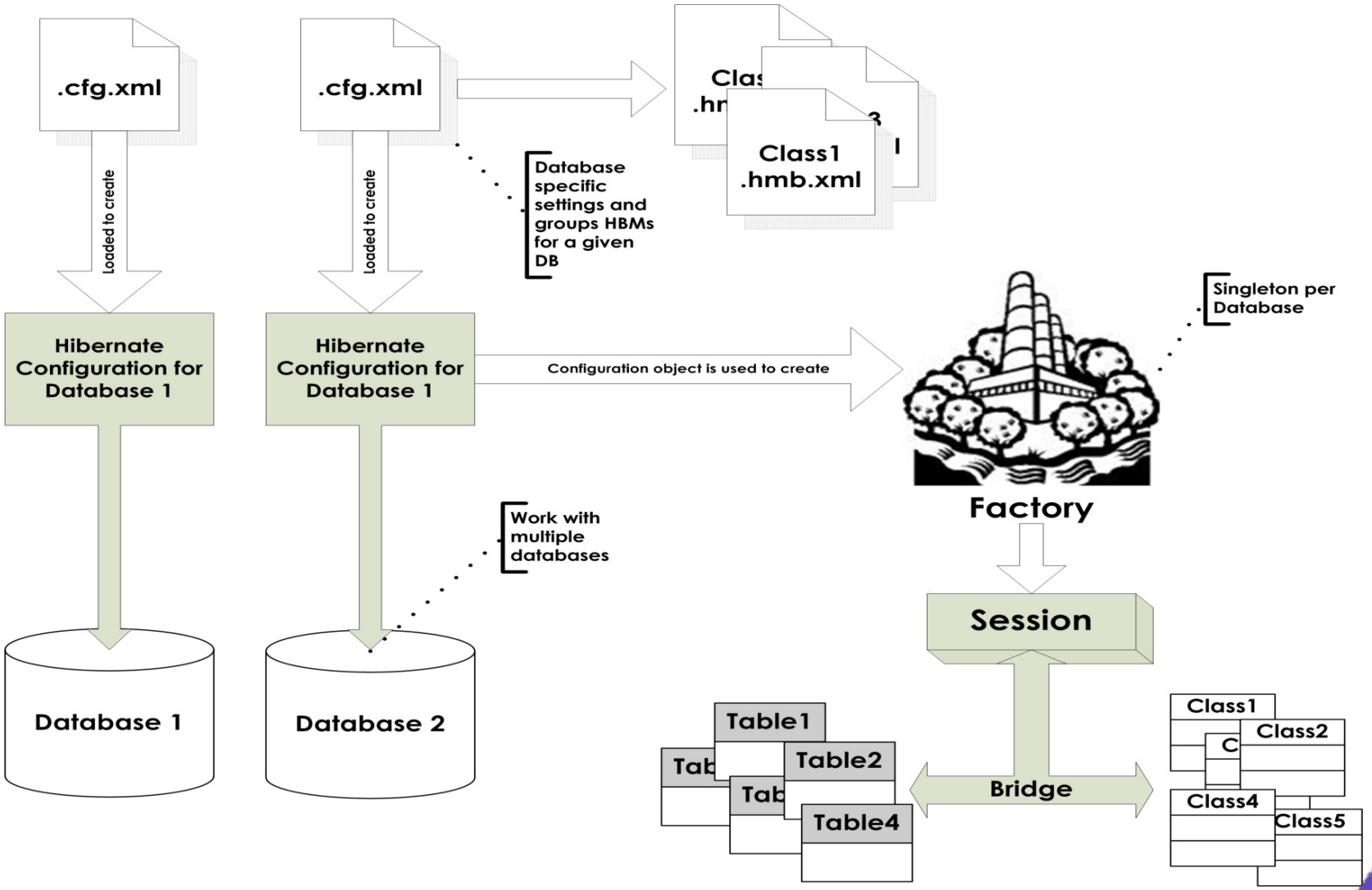
Entity Class Sample



Configuring Hibernate

- ✓ Hibernate Session Factory configured via hibernate.cfg.xml
- ✓ CFG file determines which mappings to load (.hbm.xml files)
- ✓ In your application SessionFactory is a singleton
- ✓ You request a Session as needed to work with persistent objects

The Big Picture



Working with Persistent Objects
Saving an Object

```
Address address = new Address();
...
Session session = null;
Transaction tx = null;
try {
    session = factory.openSession();
    tx = session.beginTransaction();
    session.save(address);
    tx.commit();
...
} finally {
    session.close();
}
```

- POJO is created
- Start a Session
- Transaction is started
- The object is saved
- The transaction is committed
- The session is closed

Working with Persistent Objects
Loading an Object

```
Address address = null;
Session session = null;
session = factory.openSession();
try {
    address = (Address)
        session.get(Address.class, id);
} finally {
    session.close();
}
```

- Start a Session
- POJO is loaded
- PK and Object Class are provided
- The session is closed

Working with Persistent Objects
Deleting an Object

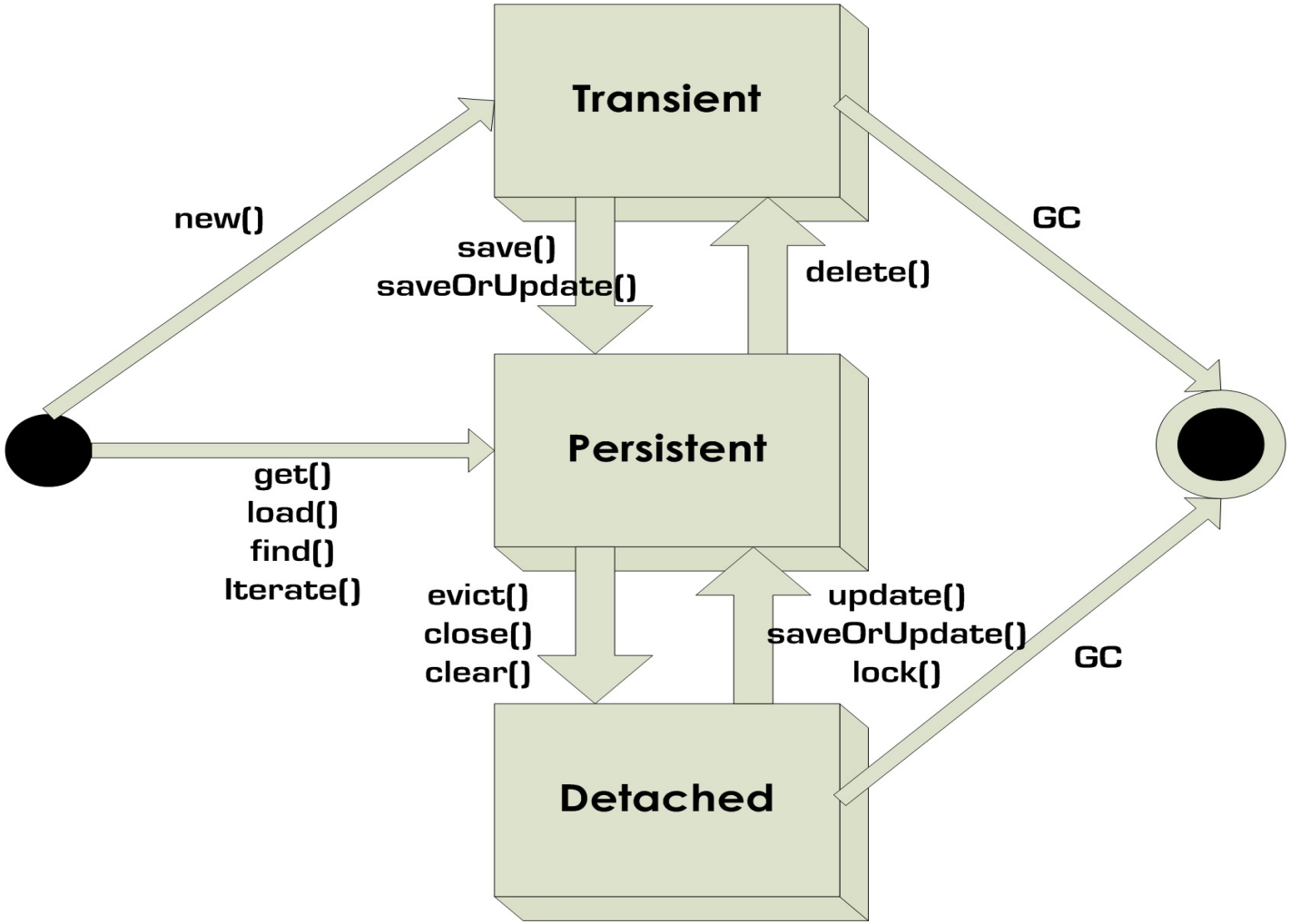
```
Session session = null;
Transaction tx = null;
try {
    session = factory.openSession();
    tx = session.beginTransaction();
    session.delete(address);
    tx.commit();
...
} finally {
    session.close();
}
```

- Start a Session
- Transaction is started
- The object is deleted
- The transaction is committed
- The session is closed

Persistence Lifecycle
Possible States

- ✓Transient
  - ✓Not Associated with a database table
  - ✓Non-transactional
- ✓Persisted
  - ✓Object with Database Identity
  - ✓Associates object with a Session
  - ✓Transactional
- ✓Detached
  - ✓no longer guaranteed to be in synch with the database

Persistence Lifecycle

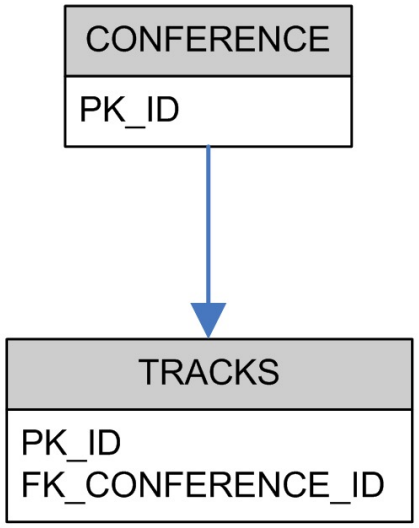


Working with Collections

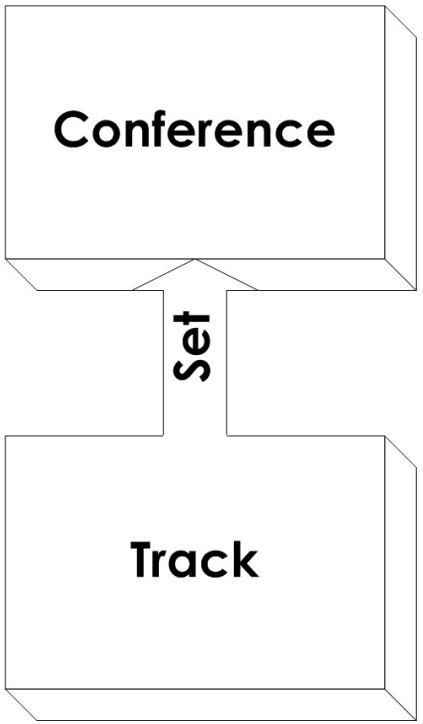
- ✓ Can represent a **parent-child/one-many** relationship using most of the available Java Collections
- ✓ Persistence by reach-ability
- ✓ No added semantics to your collections
- ✓ Available Collection mappings are
  - ✓ <set>, <list>, <map>, <bag>, <array> and <primitive-array>
- ✓ In the database mappings are defined by a **foreign key** to the owning/parent entity

Working with Collections
A simple example

Database



Java Objects



Working with Collections
A simple example

```
public class Conference implements Serializable {
    ...
    // Tracks belonging to this Conference
    private Set tracks;
    public Set getTracks () { return tracks; }
    public void setTracks (Set tracks) { this.tracks = tracks; }

    // maintain bi-directional association
    public void addTrack(Track track) {
        if (null == tracks) tracks = new HashSet();
        track.setConference(this);
        tracks.add(track);
    }
```

Working with Collections
A simple example

```
<hibernate-mapping
package=" com.ezjasoft.domain">
   <class name="Conference">
      <id column="PK_ID" name="Id“
           type="integer">
         <generator class="identity" />
      </id>
      ...
      <set name="Tracks"  inverse="true"
            cascade="all" lazy="false">
         <one-to-many class="Track" />
         <key column="FK_CONFERENCE_ID"
   />
```

- Class/Table
- How to generate PK
- Other field mappings
- Java Set for the Tracks
- Class on the many side
- Foreign Key on the many side

Association Mappings
Supported Associations

- ✓ One-to-one
  - ✓ Maintained with Foreign Keys in Database
- ✓ One-to-many / Many-to-one
  - ✓ Object on the 'one' side
  - ✓ Collection on the many 'side'
- ✓ Many-to-Many
  - ✓ Use a 'mapping' table in the database

Association Mappings
Some Details

- ✓ Inverse attribute used for bi-directional associations
- ✓ Lazy Loading
  - ✓ Configured per relationship
  - ✓ Uses dynamic proxies at Runtime
- ✓ Cascading Styles
  - ✓ **"none"** no operations are cascaded
  - ✓ **"all"** all operations are cascaded
  - ✓ Every operation in Hibernate has a corresponding cascade style

Hibernate Query Language
HQL

- ✓ An objectified version of SQL
  - ✓ Polymorphic Queries
  - ✓ Object parameters in Queries
  - ✓ Less verbose than SQL
- ✓ Doesn't hide the power of SQL
  - ✓ SQL joins, Cartesian products
  - ✓ Projections
  - ✓ Aggregation (max, avg) and grouping
  - ✓ Ordering
  - ✓ Sub-queries
  - ✓ …and more

Hibernate Query Language
HQL

- ✓ Simplest HQL Query
  - ✓ Return all Addresses

```
session = sessionFactory.openSession();
// query string – 'Address' refers to a Class not a Table
String queryString = "from Address";

// create, configure and execute the query
List addresses =
    session.createQuery(queryString).list();
```

Hibernate Query Language
HQL

- ✓ A more elaborate HQL Query
  - ✓ Return all Tracks in a Conference

```
Conference conference = ...
session = sessionFactory.openSession();
// build a query string
String queryString = "from Tracks as t where t.Conference = :conf";
 // create, configure and execute the query
List addresses = session.createQuery(queryString)
                        .setObject("conf", conference)
                        .list();
```

Hibernate Query Language
HQL Parameters

- ✓ Named parameters removed positional problems
- ✓ May occur multiple times in the same query
- ✓ Self-documenting

```
List tracks = new ArrayList();
tracks.add("JSE");
tracks.add("JEE");
Query q = session
        .createQuery("from Sessions s where s.Track in (:trackList)");
q.setParameterList("trackList", tracks);
List sessions = q.list();
```

Hibernate Query Language
HQL Parameters

- ✓ Specify bounds upon your result set
- ✓ The maximum number of rows
- ✓ The first row you want to retrieve

```
Query query = session.createQuery("from
    Users");
query.setFirstResult(50);
query.setMaxResults(100);
List someUsers = query.list();
```

Other Hibernate Features

- ✓Multiple Strategies for supporting Inheritance
- ✓Scalar Query Results
- ✓Projections
- ✓Custom SQL for Loading, Updating and Deleting
- ✓JEE/JTA Friendly
- ✓Spring Integration

Conclusions
Why should you use Hibernate?

- ✓ It's a well-designed product
- ✓ It covers 80% of your persistence needs and does a fine job on the other 20%
- ✓ You get to work with POJOs.
- ✓ Performance is good! Boils down to the generated SQL
  - ✓ If you don't like it, you can always use custom SQL

Hibernate: Object – Relational Mapping

M. En C. José Asunción Enríquez Zárate
asuncionez@gmail.com

ORACLE MySQL HIBERNATE JEDI