Hibernate
Frameworks for Data Access and SOA

Jeff Zhuk
Nick Samoylov
David Lovejoy

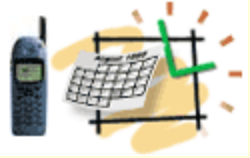
Data Access – Old Style

```
public User getUserById(int id, String psw) {
 try {
  DataManager dm = DataManager.init(); // locate DB connection pool
  Connection conn = dm.getConnection(); // get individual connection
  PreparedStatement prpStmt = conn.prepareStatement(
  "SELECT username, roleID from users where userID=? and psw=?");
  prpStmt.setInt(1,id);
  prpStmt.setString(2,psw);
  User user = new User();
  ResultSet rs = prpStmt.executeQuery();
  while (rs.next()) {
     String username = rs.getString(1);
     int roleID = rs.getInt(2);
     user.setName(username);
     user.setRoleID(roleID);
  }
  rs.close();
  prpStmt.close();
 } catch (SQLException e) {
  e.printStackTrace();
  return null;
 }
 return user;
}
```

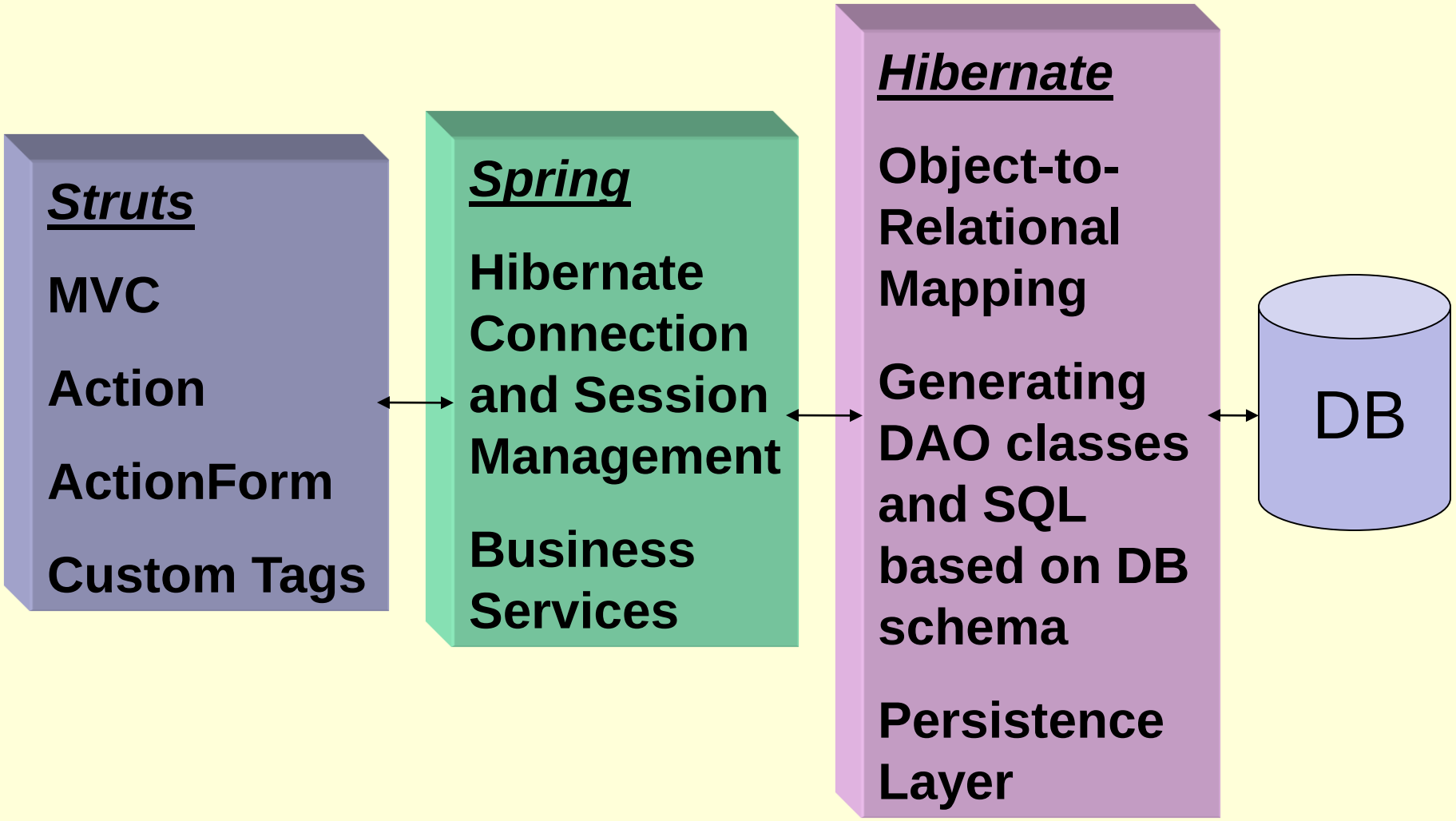
This is not the best code you can find but this is the most common approach to data access via JDBC. This code can be located in EJB or plain Java class working with data.

Note, that with this approach we must remember data types and provide multiple lines of code each time we access data.

Some of these lines are very generic and can be provided by frameworks.



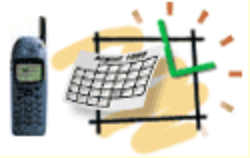
Struts – Spring - Hibernate

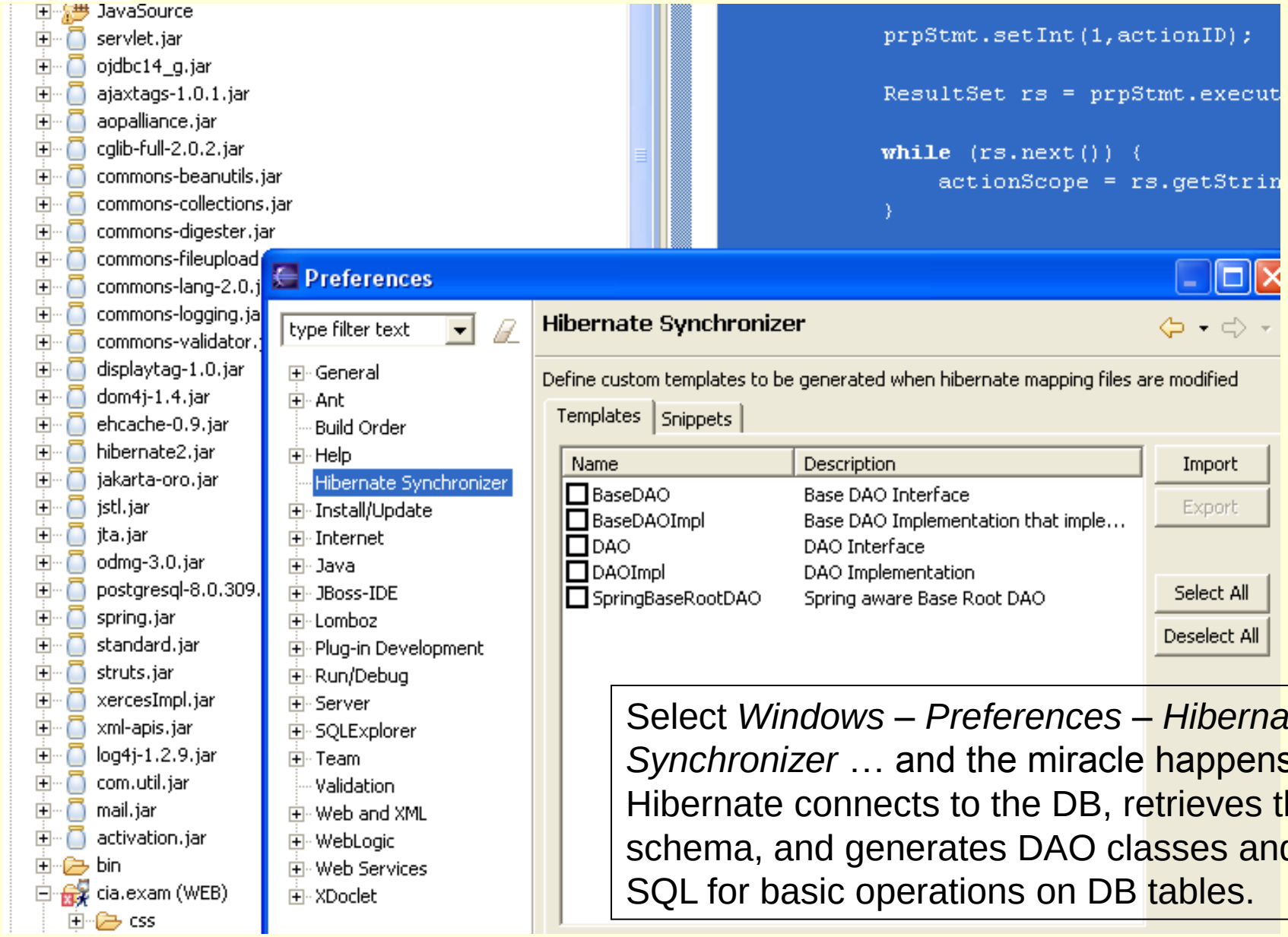


Hibernate

- Uses OO query language called HQL
- Uses objects instead of tables and fields instead of columns
- Provides object-to-relational mapping for most DBs
- Separates data layer from business logics
- Uses DB connection info to retrieve DB schema
- Generates DAO beans with data fields mapping table columns
- Generates Insert/Update/Delete/Select statements for DB tables

Hibernate Synchronizer





Select *Windows – Preferences – Hibernate Synchronizer* … and the miracle happens: Hibernate connects to the DB, retrieves the schema, and generates DAO classes and SQL for basic operations on DB tables.

Spring's Map to Hibernate

App-name.war

-WEB-INF

-- applicationContext.xml

```
<beans>
  <!-- == PERSISTENCE DEFINITIONS ===== -->

  <bean id="myDataSource"
  class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="resourceRef"><value>true</value></property>
      <property name="jndiName">
         <value>jdbc/javatest</value>
      </property>
   </bean>

  <!– Connect to Hibernate and match your "dataSource" definition -->
  <bean id="mySessionFactory"
class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
     <property name="mappingResources">
       <list>
         <value>CIAExamAnswer.hbm.xml</value>
         <value>UserRoles.hbm.xml</value>
         <value>InstructorCategory.hbm.xml</value>
       </list>
     </property>
```

Spring Maps Data Source Dialect and Provides Transaction Management for Hibernate Operations

```
<property name="hibernateProperties">
        <props>
          <prop key="hibernate.dialect">
             net.sf.hibernate.dialect.PostgreSQLDialect</prop>
          <prop key="hibernate.show_sql">true</prop>
          <prop key="hibernate.cglib.use_reflection_optimizer">true</prop>
        </props>
      </property>

      <property name="dataSource">
        <ref bean="myDataSource"/>
      </property>
    </bean>
 <!-- Transaction manager for a single Hibernate SessionFactory -->
    <bean id="myTransactionManager"
class="org.springframework.orm.hibernate.HibernateTransactionManager">
      <property name="sessionFactory">
        <ref local="mySessionFactory"/></property>
    </bean>
```

Spring and Hibernate Reduce Business Code

The sessionFactory property and the mySessionFactory bean are related in the Spring configuration file.

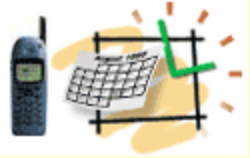
Spring creates described objects and factories that instantiate Hibernate DAO classes at run-time.

Spring simplifies the Hibernate configuration that otherwise would be stored in the *hibernate.cfg.xml* file.

The bottom line: Spring and Hibernate working together reduce your business code, ***especially when you operate with simple data records that reflect full table structure***.

Hibernate class – table mapping

```
/**
 * @hibernate.class table="GT_NOTES"
 */
public class NoteImpl implements Note {
….
}
```



Hibernate “sees” private constructor

```
/**
 * Constructor needed for hibernate.
 */
private NoteImpl()
{
    super();
}

/**
 * Constructor used to construct a note.
*/
public NoteImpl(String theNote)
{
    this.note = theNote;
}
```

Sequence based property

```
/**
 * @hibernate.id column="NOTE_ID" generator-
   class="sequence"
 * @hibernate.generator-param name="sequence"
   value="GT_NOTES_NOTE_ID_S"
 */
   public Integer getId() {
       return id;
   }
   private void setId(Integer id) {
       this.id = id;
   }
```

Object relations mapping

```
/**
 * @hibernate.many-to-one
 *     class="com.its.ebiz.ddm.model.RecipientImpl"
 *     column="RECIPIENT_ID" not-null="true"
 */
public Recipient getRecipient()
{
    return this.recipient;
}

public void setRecipient (Recipient recipient)
{
    this.recipient = recipient;
}
```

Any other convenience methods

```
public String getRecipientName()
{
    return this.recipient == (null? "" :
                this.recipient.getName());
}

public void setWhatever(Whatever w)
{
    return this.whatever = w;
}
```

Spring integration

```
<!-- Hibernate SessionFactory -->
<bean
    id="sessionFactory"class="org.springframework.orm.hibernate.Local
    SessionFactoryBean">
…
<property name="mappingResources">
<list>

    <value>com/jeppesen/ebiz/ddm/model/NoteImpl.hbm.xml</value>

    <value>com/jeppesen/ebiz/ddm/model/PkgImpl.hbm.xml</value>

 </list>
…
</bean>
```

Hibernate object mapping

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE hibernate-mapping>
- <hibernate-mapping default-cascade="none" default-access="property" auto-
      import="true">
- <class name="com.its.ebiz.ddm.model.NoteImpl" table="GT_NOTES" mutable="true"
      polymorphism="implicit" dynamic-update="false" dynamic-insert="false" batch-
      size="1" select-before-update="false" optimistic-lock="version">
- <id name="id" column="NOTE_ID" type="java.lang.Integer" unsaved-value="null">
- <generator class="sequence">
  <param name="sequence">GT_NOTES_NOTE_ID_S</param>
  </generator>
  </id>
  <many-to-one name="recipient" class="com.its.ebiz.ddm.model.RecipientImpl"
      cascade="none" outer-join="auto" update="true" insert="true"
      column="RECIPIENT_ID" not-null="true" unique="false" />
  </class>
  </hibernate-mapping>
```

Transaction support

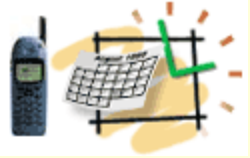
```
<!-- Transactional proxy for the Recipient primary business object -->
<bean id="domain"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionAttributes">
 <props>

  <prop key="delete*">PROPAGATION_REQUIRED</prop>

  <prop key="find*">PROPAGATION_REQUIRED,readOnly</prop>

  </props>
 </property>
</bean>
```

Application context

```
ApplicationContext context = new
    ClassPathXmlApplicationContext (
"applicationContext-test.xml");
Façade f = context.getBean(“facadeFactory”);
Whatever w = f.findWhatever();
Note n = new NoteImpl(“The note”);
f.store(n);
```

web.xml

```
<context-param>
   <param-name>
          contextConfigLocation
   </param-name>
   <param-value>
          /WEB-INF/applicationContext.xml
    </param-value>
</context-param>

<filter>
   <filter-name>hibernateFilter</filter-name>
   <filter-class>org.springframework.orm.hibernate.support.OpenSessionInViewFilter
   </filter-class>
</filter>
```



Hibernate with JTA for SOA

- Concepts
  - Persistance as a Framework for Services
  - Let's Consider RDBMS:
  - Relational Database Management Systems (RDBMs)
    - Often the platform of choice to persist application data
      - Mature, stable, portable
      - Multi user capabilities
      - Security
      - Robustness
      - Transactional (data integrity)
      - Maintain referential integrity
      - COTS/OSS Utilities and Tools for reporting and maintenance

Hibernate with JTA for SOA

- Concepts (cont.)
  - Object Oriented Programming Languages (OOPLs)
    - Often the choice for implementing SOA's.
    - Rich libraries available for web and message based service processing
    - All the benefits of an object oriented approach
      - Inheritance
        - Re-use, reduces duplication of code
      - Encapsulation
        - Security, abstraction
      - Polymorphism
        - Reduces coupling between software components
        - Makes software more extensible

Hibernate with JTA for SOA

- Concepts (cont.)
  - So, we want to use and RDBMS with OOPL… What's the problem?
    - Object-relational Impedance Mismatch

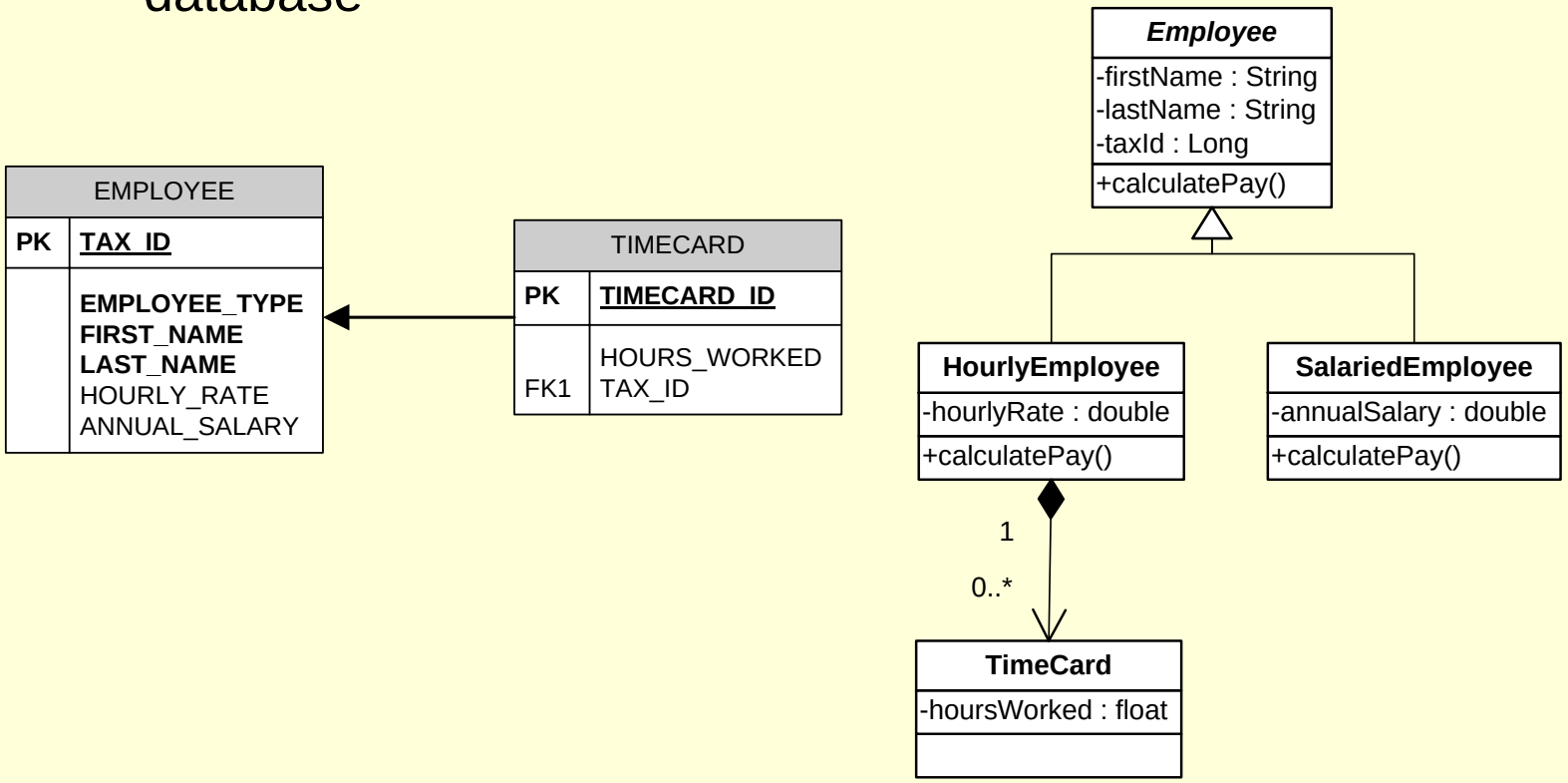
Concept	RDBMS	OO Language
Inheritance, Polymorphism	Schema Bound (All Rows in a table have the same columns)	Every object in a collection may have different attributes, and/or behaviors
Access Rules (encapsulation)	Table level permissions	Attribute level permissions
Entity Uniqueness	Rows must have identity (primary key)	Objects can typically be known by their association to other objects (no externally viewable identifiers)
Normalization	Intrinsic and required by good relational design	Often ignored by OO designs. Object graphs often result in a network database which can be extremely de-normalized
Schema Inheritance	Not supported by most RDBMS.	Intrinsic with all Object Oriented Programming Languages
Structure vs. Behavior	Efficiency, adaptability, fault-tolerance, integrity	Maintainable, understandable, extensible, reusable, safe
Data Types	Vendor specific representations	Language specific representations

Hibernate with JTA for SOA

- Concepts (cont.)
  - Object Relational Modeling (ORM)
    - Bridges the gap between object based software and relational persistence
      - Maintain data as objects in software which provides:
        - Inheritance, Encapsulation, Polymorphism
      - Persist data in a Relational Database that is useful for:
        - Reporting / Extracts
        - Ad Hoc Queries
        - Maintain Referential and Data Integrity
    - Automated (transparent) mapping of in memory objects to associated relational database tables for CRUD and search operations.
    - Allows programmers inside of code to treat the data as standard objects and maintain object oriented design principles and patterns

Hibernate with JTA for SOA

- Concepts (cont.)
  - Example: map object oriented software to a relational database



Hibernate with JTA for SOA

- **ORM Basic Concepts**
  - Mapping of database rows to objects
    - Entities and their Attributes
    - Entity Associations
      - Multiplicity
        - One-to-One, One-to-Many, Many-to-Many
      - Directional relationships
        - Ex. OrderItem ← → Order
    - Inheritance
      - Table-per-class, table-per-subclass, table-per-concrete-class
  - Mapping Metadata
    - Describes the objects and their mapping to underlying database tables
  - Type conversion
    - Automatic Type conversion
      - Ex. Varchar2 → String → Varchar2
    - User defined type conversion
  - Locking schemes
    - Optimistic / Pessimistic

Hibernate with JTA for SOA

- **ORM Basic Concepts (cont.)**
  - Lazy association loading
  - Collection classes (Maps, Sets, Ordered Sets, etc…)
  - Support Transactions (unit of work)
  - Caching
  - Queries
    - Specialized Query Languages (i.e. HQL, JPAQL)
    - Support for Native SQL
    - Named Queries
  - Encapsulation of SQL generation
    - Database platform becomes a configuration item
  - Transitive Persistence (Cascade)
  - Automatic “dirty checking”

Hibernate with JTA for SOA

- Hibernate
  - ORM solution
  - Provides persistence for Plain Old Java Objects (POJOs)
  - Persistence mechanism for Java (standalone) and J2EE (application server)
  - Supports inheritance, polymorphism, encapsulation, composition
  - Open Source (GNU Lesser General Public License)
  - Doesn't require preprocessing or extra code generation
  - Accessible through Hibernate API or Java Persistence API (EJB 3.0)

"Hibernate's goal is to relieve the developer from 95 percent of common data persistence related programming tasks, compared to manual coding with SQL and the JDBC API."

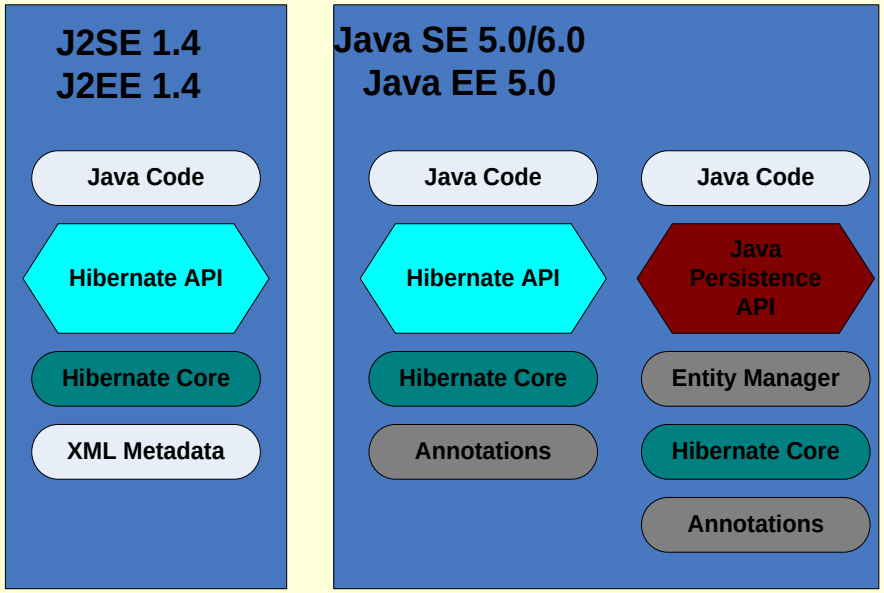
Hibernate with JTA for SOA

- Hibernate (cont.)
  - Hibernate modules

Hibernate Core Hibernate for Java, native APIs and XML mapping metadata

Hibernate Annotations Map classes with JDK 5.0 annotations

Hibernate EntityManager Standard Java Persistence API for Java SE and Java EE



Hibernate with JTA for SOA

- Hibernate (cont.)
  - Hibernate API
    - Hibernate Queries
      - Hibernate Query Language (HQL)
      - Object based, not schema based
      - Results are retrieved as objects or sets of objects
      - Support for major SQL functions and operators (i.e. left|right outer join, group by, min, max, sum, subselects, etc…)
      - Support for dynamic fetch parameters
      - Criteria based queries
      - Polymorphic queries
    - Native Queries
      - Option to write your own JDBC SQL queries
      - Hibernate still provides a translation of results into objects

Hibernate with JTA for SOA

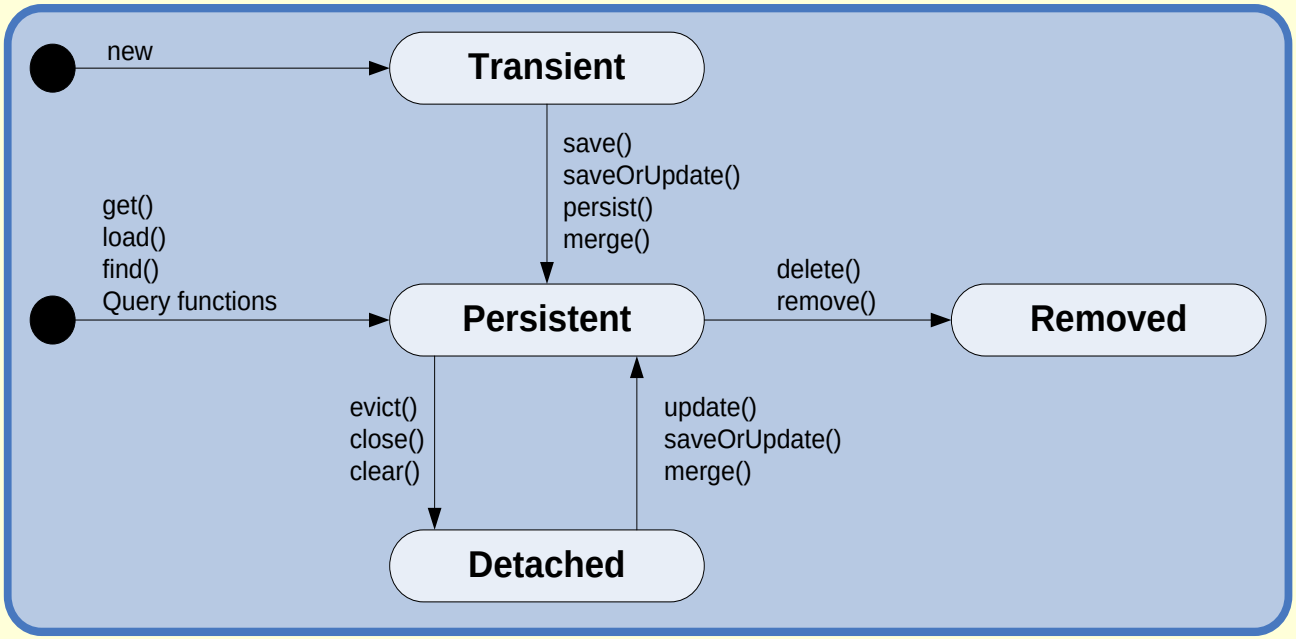
- Hibernate (cont.)
  - Hibernate API (cont.)
    - Additional features:
      - Automated support for conversion between database variants and Java datatypes
      - Support for custom data types
      - Automatic “dirty checking”
      - Supports long-lived persistence contexts through detach/reattach
      - Support for extensive subset of the Java Collections API (with indexing)
      - Inheritance (Table-per-class, table-per-subclass, table-per-concrete-class)
      - Transitive persistence (cascades)
      - Optimistic and Pessimistic locking

Hibernate with JTA for SOA

- Hibernate (cont.)
  - Hibernate API (cont.)
    - SessionFactory
      - Heavyweight object created once.
      - Reads configuration files and metadata mapping files
      - Provides access to the Hibernate Session
    - Session
      - Lightweight object used to perform most persistent operations
      - Provides Persistence Context (cache)
      - Responsible for keeping registered persistent objects in sync with the database

Hibernate with JTA for SOA

- Hibernate (cont.)
  - Hibernate API (cont.)
    - Entity States

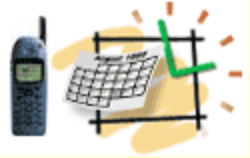


Hibernate with JTA for SOA

- EJB3.0 / JPA Overview
  - **EJB 3.0 is not EJB 2.0 (Entity Beans)**
  - Standardizes Java ORM into a common API
  - Vendors provide implementations of the API
    - Hibernate, TopLink, OpenJPA, Kodo, etc…
  - Consortium of industry leading experts contributed
  - Shows a clear influence from Spring in its use of POJO's and dependency injection
  - Hibernate's influence is even more obvious.
    - JPA API is very similar to Hibernate
    - Most features in Hibernate were incorporated into the JPA
  - You do not need an EJB container or Java EE application server in order to run applications that use JPA persistence.

Hibernate with JTA for SOA

- EJB3.0 / JPA Specifics
  - EntityManagerFactory
    - Session Factory
  - EntityManager
    - Session

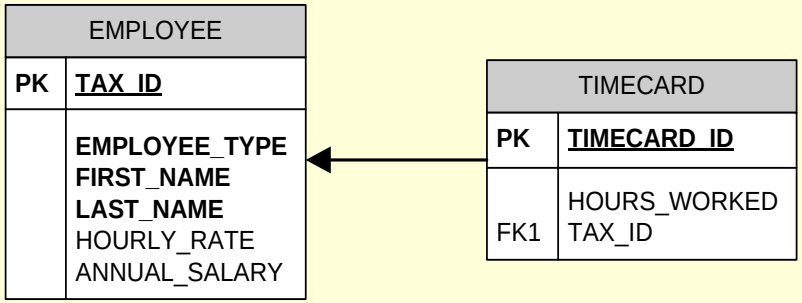


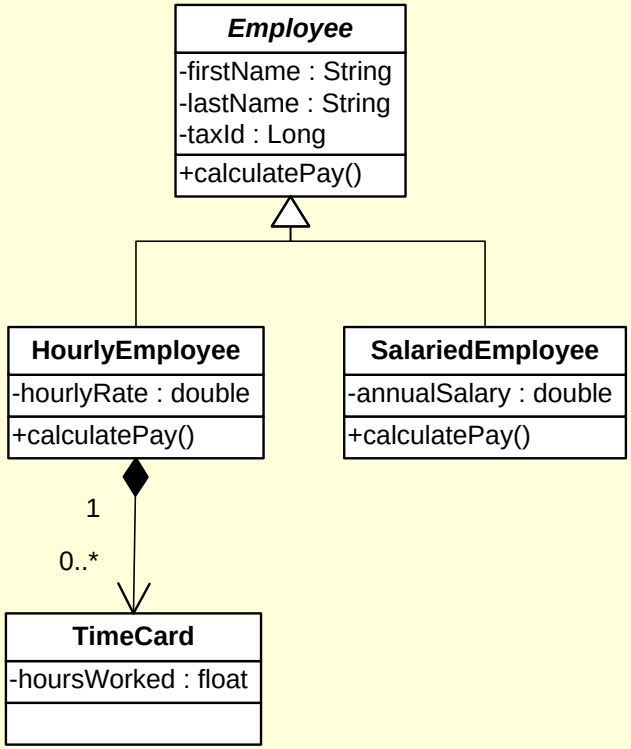
Hibernate with JTA for SOA

- EJB3.0 / JPA Specifics (cont.)
  - JPA vs. Native Hibernate
    - Configuration
      - META-INF/persistence.xml
    - Criteria Based Queries
    - Custom Data Types
    - JPA QL
      - Subset of HQL
      - All queries in JPA QL are valid in HQL
    - Optimistic Locking
  - Vendor extensions

Example

- Example





Example

```
@Entity
@Table(name='EMPLOYEE')
public abstract class Employee {

@Id
@Column(name='TAX_ID')
private Long employeeId;

@Column(name='FIRST_NAME')
private String firstName;

@Column(name='LAST_NAME')
private String lastName;

.
.
.
}
```

Example

```
@Entity
@Table(name='EMPLOYEE')
    @Inheritance( strategy = InheritanceType.SINGLE_TABLE )
•   @DiscriminatorColumn ( name='SOURCE_RQST_TYPE',
•               discriminatorType=DiscriminatorType.STRING )
public abstract class Employee {

@Id
@Column(name='TAX_ID')
private Long employeeId;

@Column(name='FIRST_NAME')
private String firstName;

@Column(name='LAST_NAME')
private String lastName;

.
.
.
}
```

Example

```
@Entity
@DiscriminatorValue( 'SAL' )
public abstract class SalariedEmployee
    extends Employee {

@Column(name='ANNUAL_SALARY')
private Double annualSalary;

.
.
.
}
```

Example

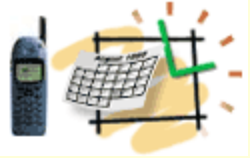
```
@Entity
@DiscriminatorValue( 'HRLY' )
public abstract class HourlyEmployee
    extends Employee {

@Column(name='HOURLY_RATE')
private Double hourlyRate;

@OneToMany( mappedBy='employee', cascade = CascadeType.ALL)
public Collection<Timecard> timecards;
.
.
.
}
```

Example

```
@Entity
@Table(name='TIMECARD')
public class Timecard {

@Id
@Column(name='TIMECARD_ID')
private Long timecardId;

@Column(name='HOURS_WORKED')
private float hoursWorked;

@ManyToOne
@JoinColumn( name='TAX_ID' )
Private Employee employee;

.
.
.
}
```

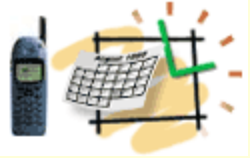
Example configuration (persistence.xml):

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd version="1.0">

    <persistence-unit name="payroll-entities" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>

        <class>com.company.payroll.Employee</class>
      <class>com.company.payroll.SalariedEmployee</class>
     <class>com.company.payroll.HourlyEmployee</class>
        <class>com.company.payroll.Timecard</class>


       <properties>
        <property name="hibernate.dialect"
   value="org.hibernate.dialect.OracleDialect"/>
        <property name="hibernate.show_sql" value="true" />
      </properties>
    </persistence-unit>
</persistence>
```



Java Code:

```
EntityManagerFactory entityFactory =
   Persistence.createEntityManagerFactory('manager');

EntityManager em = entityFactory.createEntityManager();

Employee employee =
    new HourlyEmployee(12345,'Ben','Franklin',7.55d);



em.persist(employee);
Timecard t1 = new Timecard(1, 50.00);
Timecard t2 = new Timecard(2, 80.00);

employee.addTimecard(t1);
employee.addTimecard(t2);

em.flush()
```



Java Code: Query example

```
Query aQuery = em.createQuery('from Employee where
   firstName='John'');

List<Employee> results = aQuery.getResultList();

Iterator<Employee> listIter = results.iterator();

While (listIter.hasNext()) {
    Employee e = listIter.next();

    System.out.println(e. getFirstName() +
       'earned: ' + e.calculatePay());
}
```

Hibernate with JTA for SOA

- Resources
  - **Java Persistence with Hibernate**, Gavin King and Christian Bauer, Manning Publications, C 2007
  - http://www.hibernate.org/5.html, JBoss Labs hibernate website – User / developer documentation
  - Sun Microsystems Java Persistence API website – User / developer documentation:

http://java.sun.com/javaee/technologies/persistence.jsp