The Object-Oriented Paradigm
- The world consists of objects
- So we use object-oriented languages to write applications
- We want to store some of the application objects (a.k.a. persistent objects), e.g. accounts, customers, employees
- So we use an Object Database?

The Reality of DBMS
- Relational DBMS are still predominant
  - Best performance
  - Most reliable
  - Widest support
- Bridge between OO applications and relational databases
  - Call-Level Interface (CLI) and embedded SQL
  - Object-Relational Mapping (ORM) tools

Call-Level Interface (CLI)
- Application interacts with the database through function calls
  
  ```java
  String sql = "select name from items where id = 1;"
  Connection c = DriverManager.getConnection( url );
  Statement stmt = c.createStatement();
  ResultSet rs = stmt.executeQuery( sql );
  if( rs.next() )  System.out.println( rs.getString("name") );
  ```

Embedded SQL
- SQL statements are embedded in the host language
  
  ```java
  String name;
  #sql {select name into :name from items where id = 1};
  System.out.println( name );
  ```

Employee – Application Object
- public class Employee {
  Integer id;
  String name;
  Employee supervisor;
}
Employee – Database Table

```sql
create table employees ( 
  id integer primary key,
  name varchar(255),
  supervisor integer references employees(id)
);
```

From Database to Application

◆ So how do we construct an Employee object based on the data from the database?

```
public class Employee { 
  Integer id;
  String name;
  Employee supervisor;

  public Employee( Integer id ) 
  { 
    // access database to get name and supervisor 
  }
}
```

Problems with CLI and Embedded SQL ...

◆ SQL statements are hard-coded in applications

```
public Employee( Integer id ) { 
  ...
  PreparedStatement p;
  p = connection.prepareStatement(
    "select * from employees where id = ?"
  );
  ...
}
```

... Problems with CLI and Embedded SQL ...

◆ Tedious translation between application objects and database tables

```
public Employee( Integer id ) { 
  ..
  ResultSet rs = p.executeQuery();
  if( rs.next() )
  {
    name = rs.getString("name");
  ..
}
```

... Problems with CLI and Embedded SQL

◆ Application design has to work around the limitations of relational DBMS

```
public Employee( Integer id ) { 
  ..
  ResultSet rs = p.executeQuery();
  if( rs.next() )
  {
    ..
    supervisor = ??
  }
}
```

The ORM Approach
Advantages of ORM

- Make RDBMS look like ODBMS
- Data are accessed as objects, not rows and columns
- Simplify many common operations. E.g. `System.out.println(e.supervisor.name)`
- Improve portability
  - Use an object-oriented query language (OQL)
  - Separate DB specific SQL statements from application code
- Caching

Common ORM Tools

- Java Data Object (JDO)
  - One of the Java specifications
  - Flexible persistence options: RDBMS, OODBMS, files etc.
- Hibernate
  - Most popular Java ORM tool right now
  - Persistence by RDBMS only
- Others

Hibernate Application Architecture

A Simple Hibernate Application

- Java classes
  - Employee.java
- O/R Mapping files
  - Employee.hbm.xml
- Hibernate configuration file
  - hibernate.cfg.xml
- (Optional) Logging configuration files
  - Log4j.properties
- Code to access the persistent objects
  - EmployeeTest1.java

Java Classes

- Plain Java classes (POJOs); however, it is recommended that
  - Each persistent class has an identity field
  - Each persistent class implements the Serializable interface
  - Each persistent field has a pair of getter and setter, which don’t have to be public

O/R Mapping Files

- Describe how class fields are mapped to table columns
- Three important types of elements in a a mapping file
  - `<id>`
    - when the field is of simple type
    - Association – when the field is of a class type
      - `<one-to-one>`
      - `<many-to-one>`
      - `<many-to-many>`
Hibernate Configuration Files
- Tell hibernate about the DBMS and other configuration parameters
- Either hibernate.properties or hibernate.cfg.xml or both
  - Sample files come with the downloaded Hibernate package

Access Persistent Objects
- Session
- Query
- Transaction
  - A transaction is required for updates

Hibernate Query Language (HQL)
- A query language that looks like SQL, but for accessing objects
- Automatically translated to DB-specific SQL statements
- `select e from Employee e where e.id = :id`
  - From all the Employee objects, find the one whose id matches the given value

CRUD Example
- EmployeeTest2.java
  - `load()` and `get()`
  - How does hibernate tell whether an object is new??
  - Caching and Isolation Levels

load() vs. get()
- `load()` raises an exception if an object cannot be found; `get()` would return null
- `load()` may return a proxy but `get()` never does

Caching
- Object cache and query cache
- Cache scope and cache consistency
Cache Scopes
- Session
- Process
- Cluster

Transaction Isolation Levels
- Serializable
- Read Repeatable
- Read Committed
- Read Uncommitted

Isolation Example ...

<table>
<thead>
<tr>
<th>Sells</th>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
<td></td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
<td></td>
</tr>
<tr>
<td>Sue's</td>
<td>Miller</td>
<td>3.00</td>
<td></td>
</tr>
</tbody>
</table>

- Sue is querying Sells for the highest and lowest price Joe charges.
- Joe decides to stop selling Bud and Miller, but to sell only Heineken at $3.50

Potential Problems of Concurrent Transactions
- Caused by interleaving operations
- Caused by aborted operations
- For example:
  - MAX, DEL, MIN, INS
  - MAX, DEL, INS, MIN

Caching in Hibernate
- First-level cache
  - Session scope
  - Always on (and cannot be turned off)
- Second-level cache
  - Pluggable Cache Providers
  - Process cache
    - EHCache and DSCache
  - Cluster cache
    - SwarmCache and JBossCache
Hibernate Cache Concurrency Policies

- Transactional
- Read-Write
- Non-strict Read-Write
- Read-only
- Read Repeatable
- Read Committed
- Read Uncommitted

Currency Support of Hibernate Cache Providers

<table>
<thead>
<tr>
<th>Cache Provider</th>
<th>Read-only</th>
<th>Non-strict Read-Write</th>
<th>Read-Write</th>
<th>Transactional</th>
</tr>
</thead>
<tbody>
<tr>
<td>EhCache</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CoCache</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SwarmCache</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BossCache</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

hbm2ddl

- Generate DDL statements from Java classes and mapping files
- db/hibernate.ddl – generated automatically by hbm2ddl

More About Mapping

- Basic mapping
  - <id>
  - <property>
  - Association
    - many-to-one
    - one-to-many
    - one-to-one
    - many-to-many
- Collections
- Subclasses
- Components
- Other
  - Bidirectional association

Collection of Simple Types

```java
public class Customer {
    Integer id;
    String name;
    String address;
    Set<String> phones;
}
```

Map Set of Simple Types

```xml
<set name="phones" table="phones">
    <key column="customer_id"/>
    <element type="string" column="phone"/>
</set>
```

Map Set of Simple Types

| id | customer_id | phone |
Map List of Simple Types

```xml
<list name="phones" table="phones">
  <key column="customer_id"/>
  <index column="phone_order"/>
  <element type="string" column="phone"/>
</list>
```

<table>
<thead>
<tr>
<th>customers</th>
<th>phones</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>customer_id</td>
</tr>
</tbody>
</table>

Collection of Object Types

```java
public class Account {
    Integer id;
    Double balance;
    Date createdOn;
    List<Customer> owners;
}
```

Map List of Object Types

```xml
<list name="owners" table="ownship">
  <key column="account_id"/>
  <index column="owner_order"/>
  <many-to-many class="Customer" column="customer_id"/>
</list>
```

<table>
<thead>
<tr>
<th>customers</th>
<th>ownship</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>customer_id</td>
<td>owner_order</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inheritance

```java
public class CDAccount extends Account {
    Integer term;
}
```

Map Subclass – Table Per Concrete Class

```xml
<joined-subclass name="CDAccount" table="cd_accounts">
  <key column="account_id"/>
  <property name="term"/>
</joined-subclass>
```

<table>
<thead>
<tr>
<th>accounts</th>
<th>cd_accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>balance</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>cd_accounts</td>
<td>id</td>
</tr>
<tr>
<td>accounts</td>
<td>id</td>
</tr>
</tbody>
</table>
Map Subclasses – Table Per Hierarchy

```xml
<discriminator column="account_type" type="string"/>
<subclass name="CDAccount" discriminator-value="CD">
  <property name="term"/>
</subclass>
```

```
accounts
| id | balance | created_on | term |
```

Components

```java
public class Address {
  String street, city, state, zip;
}
```

```java
public class User {
  Integer id;
  String username, password;
  Address address;
}
```

Map Components

```xml
<component name="address" class="Address">
  <property name="street"/>
  <property name="city"/>
  <property name="state"/>
  <property name="zip"/>
</component>
```

```
users
| id | ... | street | city | state | zip | ...
```

Components Inside Collection

```xml
<list name="history" table="bibtex_history">
  <key column="bibtex_id"/>
  <index column="bibtex_order"/>
  <composite-element class="BibtexEntry">
    <property name="content"/>
    <many-to-one name="editor" class="User"/>
    <property name="lastModified" column="last_modified"/>
  </composite-element>
</list>
```

Bidirectional Association

```java
public class Account {
  Integer id;
  Double balance;
  Date createdOn;
  List<Customer> owners;
}
```

```java
public class Customer {
  Integer id;
  String name;
  String address;
  Set<String> phones;
  Set<Account> accounts;
}
```

Bidirectional Association Mapping

```xml
<class name="Customer" table="customers">
  ...  
  <set name="accounts" table="ownership" inverse="true">
    <key column="customer_id"/>
    <many-to-many class="Account" column="account_id"/>
  </set>
</class>
```
O/R Mapping vs. ER-Relational Conversion

<table>
<thead>
<tr>
<th>O/R Mapping</th>
<th>ER-Relational Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Entity Set</td>
</tr>
<tr>
<td>&lt;property&gt;</td>
<td>Attribute</td>
</tr>
<tr>
<td>Association</td>
<td>Relationship</td>
</tr>
<tr>
<td>Subclass</td>
<td>Subclass</td>
</tr>
<tr>
<td>• table per concrete class</td>
<td>• OO method</td>
</tr>
<tr>
<td>• table per class hierarchy</td>
<td>• NULL method</td>
</tr>
<tr>
<td>• table per subclass</td>
<td>• ER method</td>
</tr>
</tbody>
</table>

Lazy Loading

- Hibernate is “lazy” by default
  - Account -> Customers -> Phones
- But sometimes we want to be “eager”
  - Performance optimization, i.e. reduce the number of query requests
  - Disconnected clients

Fetch Strategies

- Select and subselect
- Batch size
- Join fetch

Hibernate Support in Spring

- HibernateTemplate
  - [http://www.springframework.org/docs/api/org/springframework/orm/hibernate/HibernateTemplate.html](http://www.springframework.org/docs/api/org/springframework/orm/hibernate/HibernateTemplate.html)
- CSNS source code under
  - src/csns/model/dao/hibernate
- And much more (covered later in the lectures on Spring)

The Spring Advantage

**Without Spring**
```
Transaction tx = null;
try {
  tx = s.beginTransaction();
  s.saveOrUpdate( e );
  tx.commit();
} catch( Exception e ) {
  if( tx != null ) tx.rollback();
  e.printStackTrace();
}
```

**With Spring**
```
getHibernateTemplate().saveOrUpdate( user );
```

Hibernate Projects ...

- [http://www.hibernate.org/](http://www.hibernate.org/)
- Hibernate Core
- Hibernate Annotations
  - Use Java Annotations instead of XML to specify data mapping
- Hibernate EntityManager
  - For EJB
- Hibernate Shards
  - For using multiple databases at the same time
... Hibernate Projects

- Hibernate Validator
  - Enforces database integrity constraints both in database and in Java code using annotation
- Hibernate Search
  - Integrate Hibernate with full text search engines like Lucene
- Hibernate Tools
  - Generate Java code from database schema, Eclipse plugins, additional Ant tasks etc.
- NHibernate (Hibernate for .NET)

Readings

- Java Persistence with Hibernate by Christian Bauer and Gavin King (or Hibernate in Action by the same authors)
- Hibernate Core reference at http://www.hibernate.org
  - Chapter 3-10, 14

More Readings

- Database Systems – The Complete Book by Garcia-Molina, Ullman, and Widom
  - Chapter 2: ER Model
  - Chapter 3.2-3.3: ER to Relational Conversion
  - Chapter 4.1-4.4: OO Concepts in Databases
  - Chapter 9: OQL
  - Chapter 8.7: Transactions