Java Persistence API

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/hibernate.html

Agenda

- Java Persistence API (JPA)
- Setup and use Hibernate as a JPA provider

Java Persistence API (JPA)

Java Persistence API

- **Simplifies the development of Java EE and Java SE applications using data persistence**
- **Brings the Java community behind a single, standard persistence API**
- **Draws upon the best ideas from existing persistence technologies**
  - Hibernate, TopLink, and JDO

Java Persistence API

- **Usable both within Java SE environments as well as Java EE**
  - POJO based
  - Works with XML descriptors and annotations
- **May become part of Java SE**
  - Likely that this issue will be considered by the Java SE expert group in a future Java SE release

Main JPA Components

- **Entity Classes**
- **Entity Manager**
  - Persistence Context
- **EntityManagerFactory**
- **EntityTransaction**
- **Persistence Unit**
  - persistence.xml
- **Java Persistence Query Language (JPAQL)**
  - Query

Persistence Unit

- **Defines all entity classes that are managed by JPA**
- **Identified in the persistence.xml configuration file**
- **Entity classes and configuration files are packaged together**
  - JAR or directory that contains persistence.xml is called the root of the persistence unit
  - Needs to be inside a META-INF directory
    - Whether or not inside a jar

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="BankingApp">
  <provider>
    org.hibernate.ejb.HibernatePersistence
  </provider>
  <mapping-file>orm.xml</mapping-file>
  <class>courses.hibernate.vo.Account</class>
  <class>courses.hibernate.vo.AccountOwner</class>
  <class>courses.hibernate.vo.AccountTransaction</class>
  <class>courses.hibernate.vo.EBill</class>
  <class>courses.hibernate.vo.EBiller</class>
  ...
```

persistence.xml

```
  ...
  <properties>
    <!-- VENDOR SPECIFIC TAGS -->
    <property name="hibernate.connection.driver_class"
       value="org.apache.derby.jdbc.ClientDriver"/>
    <property name="hibernate.connection.url"
       value="jdbc:derby://localhost:1527/LECTURE10"/>
    <property name="hibernate.connection.username"
       value="lecture10"/>
    <property name="hibernate.connection.password"
       value="lecture10"/>
    <property name="hibernate.dialect"
       value="org.hibernate.dialect.DerbyDialect"/>
    <property name="hibernate.show_sql"
       value="true"/>
  </properties>
</persistence-unit>
</persistence>
```

Auto Entity Detection

- **JPA provides for auto detection**
  - No need to list individual Entity classes in persistence.xml. Looks for annotated classes and mapping files
  - Specification requires use of <class> tags in non–EE environment, but Hibernate supports the functionality in both
  - Does NOT work with non–JPA Hibernate

```
<persistence-unit name="BankingApp">
  ...
  <property
    name="hibernate.archive.autodetection"
    value="class, hbm*"/>
  ...
</persistence-unit>`
```

Enabled by default

Entity Classes

- **Managed objects mapped in one of two ways**
  - Described in orm.xml mapping file
  - Marked with annotations in individual classes
    - Identified as managed with @Entity
    - Primary key identified through the @Id
- **Contains persistent fields or properties**
  - Attributes accessed through getters/setters are 'properties'
  - Directly accessed attributes are referred to as 'fields'
  - Can not combine fields and properties in a single entity
  - Must define ALL attributes in your entity, even if they're not persisted
    - Mark as 'transient' if attribute is not managed
- **Collection-valued persistent fields and properties must use the supported Java collection interfaces**

orm.xml Mapping File

```
<entity-mappings
  xmlns="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
  "http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
  version="1.0">

  <persistence-unit-metadata>
    <!-- identifies the orm.xml as the only source
         for class definition, telling the engine
         to ignore annotations in classes -->
    <xml-mapping-metadata-complete/>

    <persistence-unit-defaults>
      <cascade-persist/>
    </persistence-unit-defaults>
  </persistence-unit-metadata>
  ...
```

Set any defaults across the persistence unit entities

orm.xml Mapping File

```
...
<package>courses.hibernate.vo</package>
<entity class="Account" access="FIELD">
  <table name="ACCOUNT" />
  <attributes>
    <id name="accountId">
      <column name="ACCOUNT_ID" />
        <generated-value strategy="AUTO" />
    </id>
    <basic name="balance" optional="false">
      <column name="BALANCE" />
    </basic>
    <version name="version">
      <column name="VERSION" />
    </version>
  <attributes>
</entity>
</entity-mappings>
```

Notice, no type definitions!

Annotations: Property Access

```
@Entity
public class Account {
  private long accountId;
  ...

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="ACCOUNT_ID")
  public long getAccountId() {...}

  public void setAccountId(long newId) {...}

  ...
}
```

Account Entity: Field Access

```
@Entity
public class Account {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @Column(name="ACCOUNT_ID")
  private long accountId;
  ...

  public long getAccountId() {...}

  public void setAccountId(long newId) {...}

  ...
}
```

EntityManagerFactory

- **Used to create EntityManager in JavaSE environment**
  - Similar to Hibernate SessionFactory
- **Created through a static method on Persistence**

```
EntityManagerFactory emf =
   Persistence
      .createEntityManagerFactory("BankingApp");
```

Remember the name
of the persistence unit

EntityManager

- Creates and removes persistent entity instances
- Finds entities by their primary key
- Allows for data querying
- Interacts with the persistence context

EntityManager

- clear() // clears the context
- close() // closes the manager
- contains() // checks for existing object
- createNamedQuery() // create named query
- createNativeQuery() // create SQL query
- getTransaction() // returns the current transaction
- lock() // locks an object
- persist() // makes an object persisten
- refresh() // refreshes an object from the database
- remove() // deletes an object from the database
- find() // retrieves an object from the database
- setFlushMode() // when data is flushed from buffer

Application Managed EntityManager

- Created and destroyed explicitly by the application
- Created through EntityManagerFactory

```
EntityManagerFactory emf =
  Persistence
    .createEntityManagerFactory("BankingApp");

EntityManager em = emf.createEntityManager();
```

Save an Entity

```
public void saveAccount(Account account) {
  EntityManager em =
      JPAUtil.getEntityManager();
  em.persist(account);
}
```

Remove an Entity

- **Can not delete objects in a detached state**
  - Must programmatically ‘merge’ before calling ‘remove’

```
public void deleteAccount(Account account) {
  EntityManager em = JPAUtil.getEntityManager();
  account = em.getReference(Account.class,
                            account.getAccountId());
  em.remove(account);
}
```

Find an Entity

- No need to cast when using JPA methods

```
public Account getAccount(int accountId) {
  EntityManager em =
     JPAUtil.getEntityManager();
  Account account =
     em.find(Account.class, accountId);
  return account;
}
```

Get a Reference to an Entity

- Lazily loads using a proxy

```
public Account getAccount(int accountId) {
  EntityManager em =
    JPAUtil.getEntityManager();
  Account account =
    em.getReference(Account.class, accountId);
  return account;
}
```

Associations

- **Associations realized through orm.xml mapping file or multiplicity annotations**
  - javax.persistence.OneToOne
  - javax.persistence.OneToMany
  - javax.persistence.ManyToOne
  - javax.persistence.ManyToMany
- **Bidirectionality defined as attributes on the annotations**
  - "mappedBy" on the inverse side
    - Think "inverse=true"
  - The many side of many-to-one bidirectional relationships may not define the mappedBy attribute
- **No ID Bag support**
  - Supports M:M, but the relationship table can not have its own primary key
  - Must use an intermediate class using two 1:M

orm.xml Mapping File

```
<entity class="Account" access="FIELD">
  ...
  <attributes>
    <id name="accountId">
      <column name="ACCOUNT_ID" />
      <generated-value strategy="AUTO" />
    </id>
    <basic name="balance" optional="false">
      <column name="BALANCE" />
    </basic>
    <version name="version">
      <column name="VERSION" />
    </version>
    <one-to-many name="ebills" mapped-by="account">
      <join-column name="ACCOUNT_ID" />
    </one-to-many>
  </attributes>
</entity>
```

Indicates the attribute on the corresponding association. i.e. each ebill in the set has an attribute called 'account'

Bidirectional Association

```
@Entity
public class Account {
   @OneToMany(mappedBy="account")
   private Set ebills;
   ...
}

@Entity
public class EBill {
   @ManyToOne
   @JoinColumn(name="ACCOUNT_ID")
   private Account account
   ...
}
```

Indicates the attribute on the corresponding association. i.e. each ebill in the set has an attribute called 'account'

Cascading

- **Achieved through the "cascade" attribute on the multiplicity annotation**
- **Multiple cascading options**
  - Persist
    - Does not cascade detached or transient objects!
  - Merge
  - Remove
  - Refresh
  - All
- **Does <u>not</u> currently provide these Hibernate additional cascading options**
  - save-update
  - delete
  - lock
  - evict
  - delete-orphan

Cascade with Annotation

```
@Entity
public class Account {
   @OneToMany(mappedBy="account",
                  cascade="CascadeType.REMOVE")
   private Set ebills;
   ...
}
```

Inheritance

- **Three ways of handling inheritance**
  - Single table per class hierarchy
    - InheritanceType.SINGLE_TABLE
  - Table per concrete entity class
    - InheritanceType.TABLE_PER_CLASS
  - "join" strategy, where fields or properties that are specific to a subclass are mapped to a different table than the fields or properties that are common to the parent class
    - InheritanceType.JOINED

Single table per class hierarchy

- **Default strategy**
  - Used if the @Inheritance annotation is not specified on the root class of the entity hierarchy
- **Table has a discriminator column to identify subclass type**
  - Specified by using @DiscriminatorColumn
    - Each entity in the hierarchy is given a unique value to store in this column
    - Can contain the following attributes
      - name
      - columnDefinition
      - discriminatorType
        - » String, Char, Integer

SINGLE_TABLE with Annotations

```
@Entity
@Table(name = "ACCOUNT")
@Inheritance(strategy =
  InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "ACCOUNT_TYPE",
  discriminatorType = DiscriminatorType.STRING)
public class Account {
  @Id
  long accountId;
  ...
}

@Entity
@DiscriminatorValue("CHECKING")
public class CheckingAccount extends Account {
  String checkStyle;
  ...
}
```

Table per concrete entity class

- **One table for each concrete subclass**
- **Support for this strategy is optional, and may not be supported by all Java Persistence API providers**
  - Default Java Persistence API provider in the Application Server does not support this strategy
    - TopLink

“join” strategy

- **Super class has a table, and each subclass has a separate table containing its specific fields**
- **Some Java Persistence API providers require a discriminator column in the table that corresponds to the root entity**
  - Including default provider in the Application Server

“join” strategy

```
@Entity
@Inheritance(strategy=JOINED)
@Table(name = "ACCOUNT")
@DiscriminatorColumn(name = "ACCOUNT_TYPE",
    discriminatorType = DiscriminatorType.STRING,
    length = 10)
public class Account {
  @Id
  long accountId;
  ...
}

@Entity
@Table(name = "CHECKING_ACCOUNT")
@DiscriminatorValue("CHECKING")
public class CheckingAccount extends Account {
  String checkStyle;
  ...
}
```

Mapped Super Class

- **Not quite ‘implicit polymorphism’, but similar**
- **Persist super class attributes in subclasses**
  - Mark super class with the "MappedSuperclass" annotation
  - Data inherited from super classes that are not marked will NOT BE PERSISTED
- **Mapped super classes are NOT QUERYABLE**

Mapped Super Class

```
@MappedSuperclass
public class Account {
  @Id
  long accountId;
  Date creationDate;
  ...
}

@Entity
@Table(name = "CHECKING_ACCOUNT")
public class CheckingAccount extends Account {
  String checkStyle;
  ...
}
```

JPA Benefits

- **Automatic scanning of deployed metadata**
- **Standardized configuration**
  - Persistence Unit
- **Standardized data access code, lifecycle, and querying capability that is fully portable**
- **Can override annotations with descriptor file**

JPA Disadvantages

- **Though standard interfaces are nice, some-what lenient spec may present gaps when switching vendor implementations**
  - Not all inheritance strategies supported
  - 'Standardized' descriptor file is basically a wrapper around vendor specific implementations
- **Missing some beneficial aspects from Hibernate**
  - Query by Example, Query by Criteria (expected later)
  - EntityManager propagation across methods/objects
  - Collection Filters
  - 2nd level Cache
  - Other minor items that developers may come to rely on
    - More-so than with most vendor-specific implementations, the temptation is to use the vendor-specific features to fill the gap – but then, no longer portable

JPA More Information

- **Oracle JPA Tutorial**
  - http://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html
- **Hibernate Documentation**
  - jpa.hibernate.org

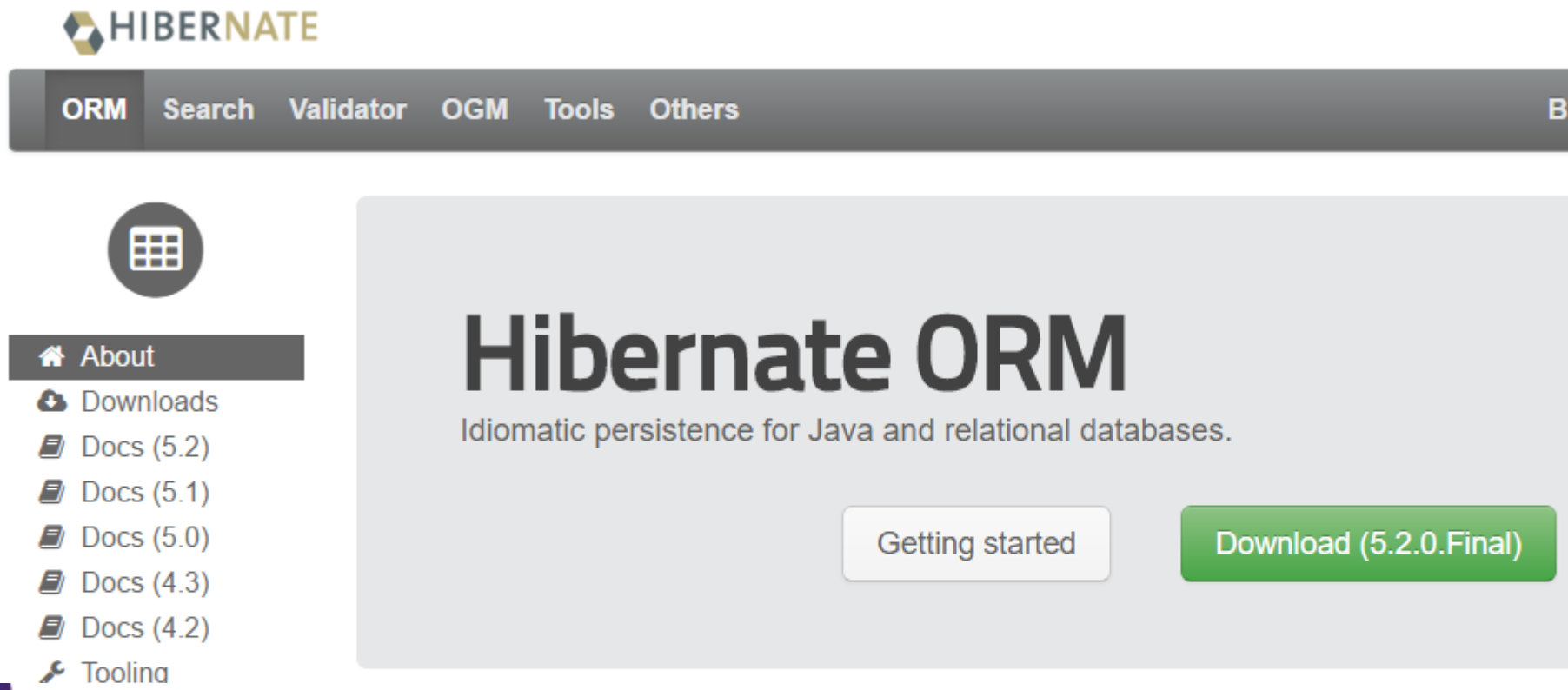
Installation

JPA with Hibernate

- **Additional jar required for compile time**
  - javaee.zip
    - Standard jar, downloadable from Java site

Hibernate – Step 1

- http://www.hibernate.org
- Click on ‘Downloads’ link

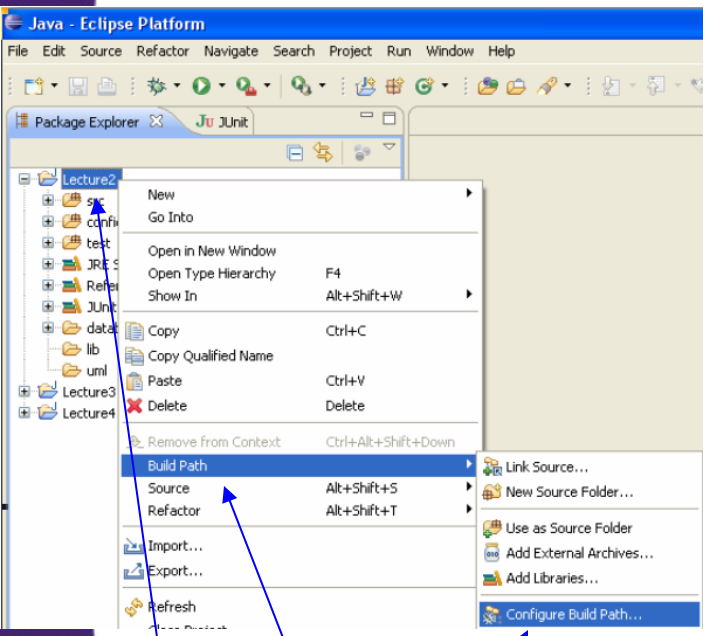


Hibernate – Step 2

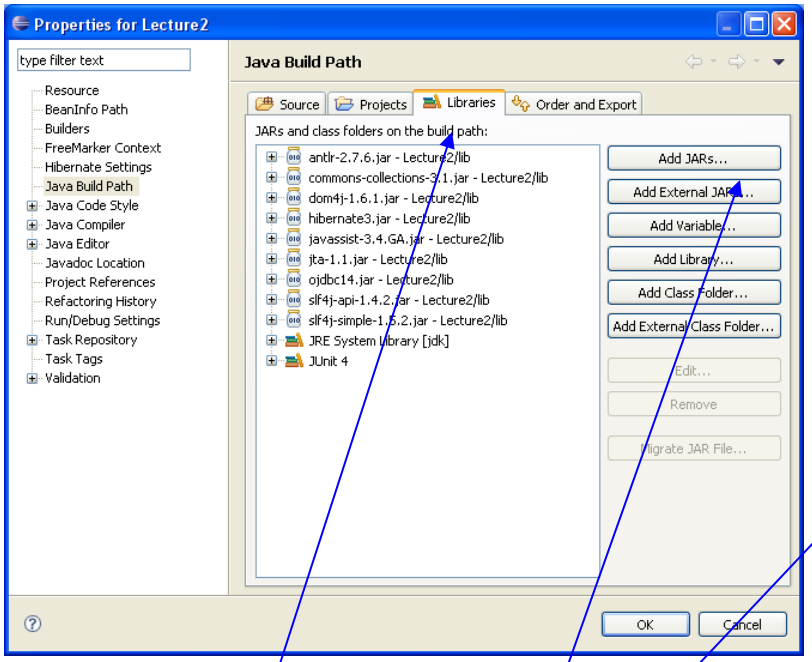
- **Unzip the Download**
  - hibernate-distribution-5.2.0.GA-dist.zip
  - Copy jars from locations under root of zip
    - hibernate5.jar
    - hibernate-distribution-5.2.0.GA/lib/required
  - Drop the jars from into the lib directory of your project (or other location you can add to your projects classpath)
- **Obtain a Simple Logging Façade for Java (SLF4J) Implementation**
  - http://www.slf4j.org/download.html
    - slf4j-1.7.21.zip
  - Unzip and copy slf4j-1.7.21.jar into lib directory of your project
    - slf4j-1.7.21.jar under root directory of download

Hibernate – Step 3

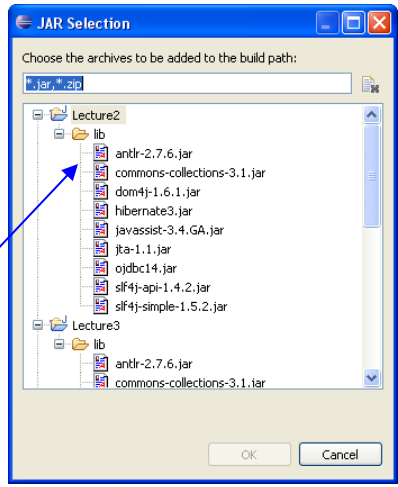
- **Within Eclipse**



Right click on project to get to "Configure Build Path"



Under the "Libraries" tab, click "Add JARs" to add the Hibernate jars to the project



JavaDB Configuration

- **JavaDB is a version of Derby that comes packaged with Java**
- **Configuration – set environment variables**
  - DERBY_HOME
    - Value should be location of JavaDB root directory
    - Example: C:\Program Files\Sun\JavaDB
  - PATH
    - Append JavaDB bin directory to existing PATH variable
    - Example: C:\Program Files\Sun\JavaDB\bin

JavaDB Configuration

- **Start Server by calling startNetworkServer script**

```
C:\WINDOWS\system32\cmd.exe - startNetworkServer
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\Sun\JavaDB\bin>startNetworkServer
Security manager installed using the Basic server security policy.
Apache Derby Network Server - 10.4.1.3 - (648739) started and ready to accept co
nnections on port 1527 at 2008-11-01 17:59:21.981 GMT
```

- **Stop Server by calling stopNetworkServer script (in another window)**

Wrap-up

Summary

- **In this lecture, we:**
  - Walked through the main components of JPA
  - Pointed out its advantages and disadvantages
  - Setup and configured Hibernate to serve as our JPA providers

Questions?