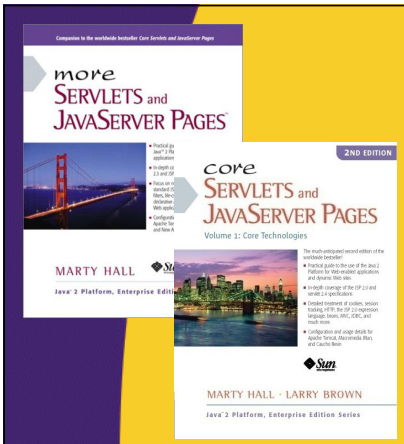
Java Persistence API

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



Topics in This Section

- Become acquainted with the Java Persistence API (JPA)
- Compare and contrast Hibernate and JPA
- Learn how to setup and use Hibernate as a JPA provider

Java Persistence API (JPA)

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

Java Persistence API

- **Developed as part of Java Specification Request (JSR) 220**
  - Original goal to simplify EJB CMP entity beans
- **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
- **Existing EJB CMP applications continue to work unchanged**
- **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

Hibernate vs. JPA Components

JPA	Hibernate
Entity Classes	Persistent Classes
EntityManagerFactory	SessionFactory
EntityManager	Session
Persistence	Configuration
EntityTransaction	Transaction
Query	Query
Persistence Unit	Hibernate Config

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
  - The JAR or directory that contains the 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="oracle.jdbc.driver.OracleDriver"/>
    <property name="hibernate.connection.url"
       value="jdbc:oracle:thin:@localhost:1521:XE"/>
    <property name="hibernate.connection.username"
       value="lecture10"/>
    <property name="hibernate.connection.password"
       value="lecture10"/>
    <property name="hibernate.dialect"
       value="org.hibernate.dialect.Oracle10gDialect"/>
    <property name="hibernate.show_sql"
       value="true"/>
  </properties>
</persistence-unit>
</persistence>
```

persistence.xml: Pass-Through

- Can satisfy the persistence.xml requirement with a pass through to an existing Hibernate configuration file

```
<persistence-unit name="BankingApp">
  <properties>
      <property name="hibernate.ejb.cfgfile"
                value="/hibernate.cfg.xml"/>
    </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 the 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**

- ***Similar to Hibernate Session***

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() // like Hibernate, but missing MANUAL

Application Managed EntityManager

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

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

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

Container Managed EntityManager

- Used with Enterprise Java Beans
- Automatically propagated to all application components within a single Java Transaction API (JTA) transaction
  - Need to identify data source in persistence.xml file
- Injected into classes with @PersistenceContext

Annotate an Entity Manager

```
public class AccountSessionBean {

@PersistenceContext
EntityManager em;

  public Account getAccount(int accountId){
    Account account =
      em.find(Account.class, accountId);

    return account;
  }
}
```

persistence.xml: Data Source

```
...
<properties>
  <jta-data-source>java:/Lecture10DS</jta-data-source>
  <!-- VENDOR SPECIFIC TAGS -->
  <property name="hibernate.connection.driver_class"
    value="oracle.jdbc.driver.OracleDriver"/>
  <property name="hibernate.connection.url"
    value="jdbc:oracle:thin:@localhost:1521:XE"/>
  <property name="hibernate.connection.username"
    value="lecture10"/>
  <property name="hibernate.connection.password"
    value="lecture10"/>
  <property name="hibernate.dialect"
    value="org.hibernate.dialect.Oracle10gDialect"/>
  <property name="hibernate.show_sql"
    value="true"/>
</properties>
...
```

Inject an EntityManagerFactory

- Inject an EntityManagerFactory into your EJB for more manual control

```
public class AccountSessionBean {

  @PersitenceUnit(unitName="BankApp")
  EntityManagerFactory emf;

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

Save an Entity in Java SE

```
public void saveAccount(Account account) {

  EntityManager em =
    JPAUtil.getEntityManager();

  EntityTransaction tx = em.getTransaction();
  tx.begin();

  em.persist(account);

  tx.commit();
  em.close();
}
```

Remove an Entity using BMP

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

```
@PersitenceUnit(unitName="BankApp")
EntityManagerFactory emf;
public void deleteAccount(Account account) {
  EntityManager em = emf.createEntityManager();
  EntityTransaction tx = em.getTransaction();
  tx.begin();

  em.remove(account);

  tx.commit();
  em.close();
}
```

Find an Entity in CMT

- **No need to cast when using JPA methods**
  - Hibernate needs to support older jdks

```
@PersistenceContext
EntityManager em;
public Account getAccount(int accountId) {
  Account account =
    em.find(Account.class, accountId);

  return account;
}
```

Get a Reference to an Entity

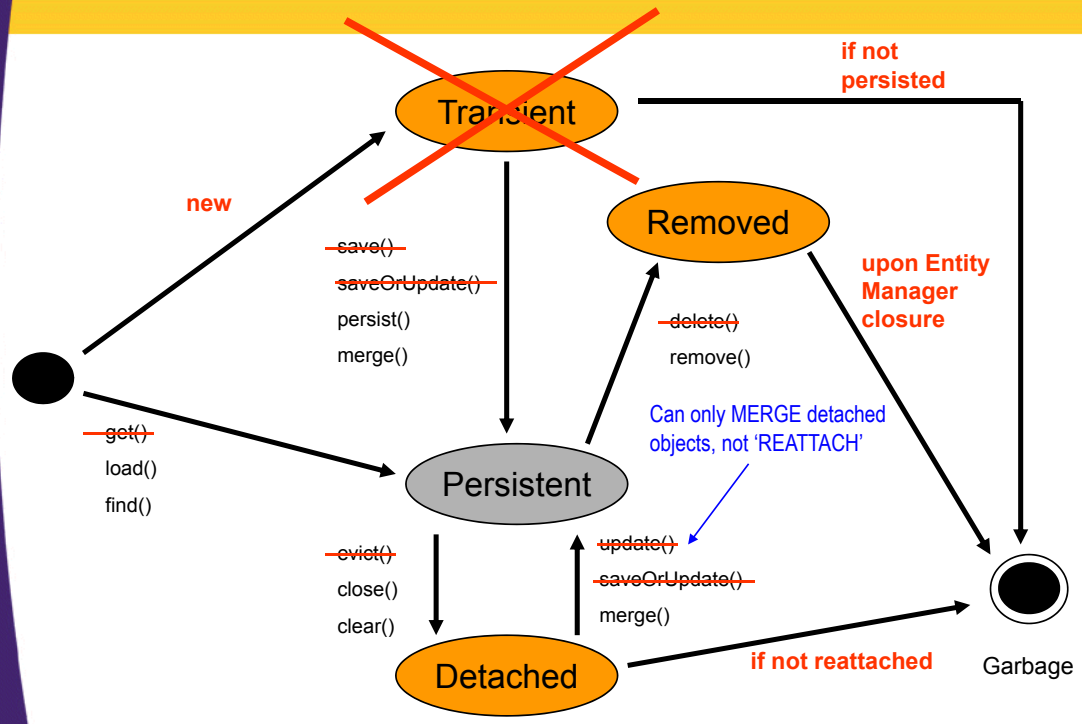
- **Similar to 'load()' method in Hibernate**
  - Lazily loads using a proxy

```
@PersistenceContext
EntityManager em;

public Account getAccount(int accountId) {
  Account account =
    em.getReference(Account.class, accountId);

  return account;
}
```

JPA Lifecycle



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 in 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" />
      <cascade>
        <cascade-remove />
      </cascade>
    </one-to-many>
  </attributes>
</entity>
```

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
- **Missing Hibernate's Implicit Polymorphism**

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 in Mapping File

```
...
<package>courses.hibernate.vo</package>
<entity class="Account" access="FIELD">
  <table name="ACCOUNT" />
  <inheritance strategy="SINGLE_TABLE" />
  <discriminator-column name="ACCOUNT_TYPE"
    discriminator-type="STRING" />
  <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>
...
```

SINGLE_TABLE in Mapping File

```
  ...
  <entity class="CheckingAccount" access="PROPERTY">
    <discriminator-value>CHECKING</discriminator-value>
    <attributes>
      <basic name="checkStyle" optional="false">
        <column name="CHECK_STYLE" />
      </basic>
    </attributes>
  </entity>
  ...
</entity-mappings>
```

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**
  - The 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 the 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;
  ...
}
```

Conversations in Java SE

- **In Hibernate, accomplished by using the same Hibernate Session throughout the request**
  - session.getCurrentSession();
- **No equivalent in JPA**
  - Always have to call emf.createEntityManager();
- **Troublesome if a request spans across multiple application DAO methods**
  - Each method obtains an EntityManager and commits/closes

Potential Solutions

- **Create the EntityManager at the instance level of the DAO**
  - Doesn't solve cross–DAO situations
- **Instantiate a single EntityManager in your service layer and pass it to called DAOs**
  - Similar to passing a Connection around
  - Create, commit, and close your EntityManager in your service
- **Simulate the Hibernate strategy yourself**
  - Create an EntityManager instance and place it on a ThreadLocal variable to use across the request

JPA Query Language

- **Subset of Hibernate Query Language**
  - Same syntax
- **Provides the @NamedQuery and @NamedNativeQuery annotations**
- **Does not support the following:**
  - Updating the version of an entity with the 'versioning' keyword
  - Some batch functionality
    - "insert...select" statements
    - ScrollableResults with cursors
  - Additional syntactical functions available in HQL

Named Query Annotations

```
import javax.persistence.*;

@NamedQueries( {
  @NamedQuery(
    name = "getAllAccounts"
    query = "from Account")

  @NamedQuery(
    name = "getAccountByBalance"
    query = "from Account where
             balance = :balance")
})
```

Hibernate Functions not in JPA

- **BIT_LENGTH(s)**
  - Returns the number of bits in S
- **CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP()**
  - Returns the current value of the database machine
- **SECOND(d), MINUTE(d), HOUR(d), DAY(d), MONTH(d), YEAR(d)**
  - Extracts the time and date from a temporal argument
- **CAST(t as Type)**
  - Casts a given type t to a Hibernate Type
- **INDEX(joinedCollection)**
  - Returns the index of a joined collection element
- **MINELEMENT(c), MAXELEMENT(c),MININDEX(c), MAXINDEX(c), ELEMENTS(c), INDICES(c)**
  - Returns an element or index of an indexed collection

JPA Listeners and Callbacks

- **Similar to Hibernate Interceptors – actions are fired on certain events**
  - @PostLoad
    - Executes after an entity is loaded
      - find(); getReference(); refresh();
  - @PrePersist; @PostPersist
    - Executes immediately when persist() is called, and after the database insert respectively
  - @PreUpdate; @PostUpdate
    - Executes immediately before and after flushing, and only for 'dirty' objects
  - @PreRemove; @PostRemove
    - Executes immediately when remove() is called or removed via a cascade, and after the database delete respectively

Creating a Listener

```
import javax.persistence.*;

// no special interface or superclass
public class AccountListener {
  @PostPersist
  public void accountCreation(Object entity) {
    logger.info("Account Created: " + entity);
  }
}
```

Assigning the Listener Callback

```
import javax.persistence.*;

@Entity
@EntityListeners(AccountListener.class)
public class Account {

  public void saveAccount(Account account) {
    ...
  }
}
```

JPA with Hibernate

- **Does not come with default Hibernate distribution**
- **Additional jar required for compile time**
  - javaee.zip
    - Standard jar, downloadable from Java site
- **Also need to download Hibernate implementation**
  - hibernate-entitymanager-3.4.0.ja.zip
  - Contains additional required jars
    - hibernate-entitymanager.jar
    - hibernate-annotations.jar
    - hibernate-commons-annotations.jar

Provided by JBoss server, but required for JavaSE runtime environment

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 there to use the vendor-specific features to fill the gap – but then, no longer portable

JPA More Information

- **JSR 220 Specification**
  - http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html
- **Sun JPA Tutorial**
  - http://java.sun.com/javaee/5/docs/tutorial/doc/bnbpy.html
- **Hibernate Documentation**
  - jpa.hibernate.org/
- **Oracle Technology Network**
  - http://www.oracle.com/technology/products/ias/toplink/jpa/index.html

Wrap-up

Summary

- **In this lecture, we:**
  - Learned about the JSR-220 intentions, and how it came about to develop the Java Persistence API Specification
  - Walked through the main components of JPA
  - Pointed out its advantages and disadvantages when compared to a vendor specific implementation, like Hibernate
  - Setup and configured Hibernate to serve as our JPA providers

Preview of Next Sections

- **That's all folks!**

Questions?