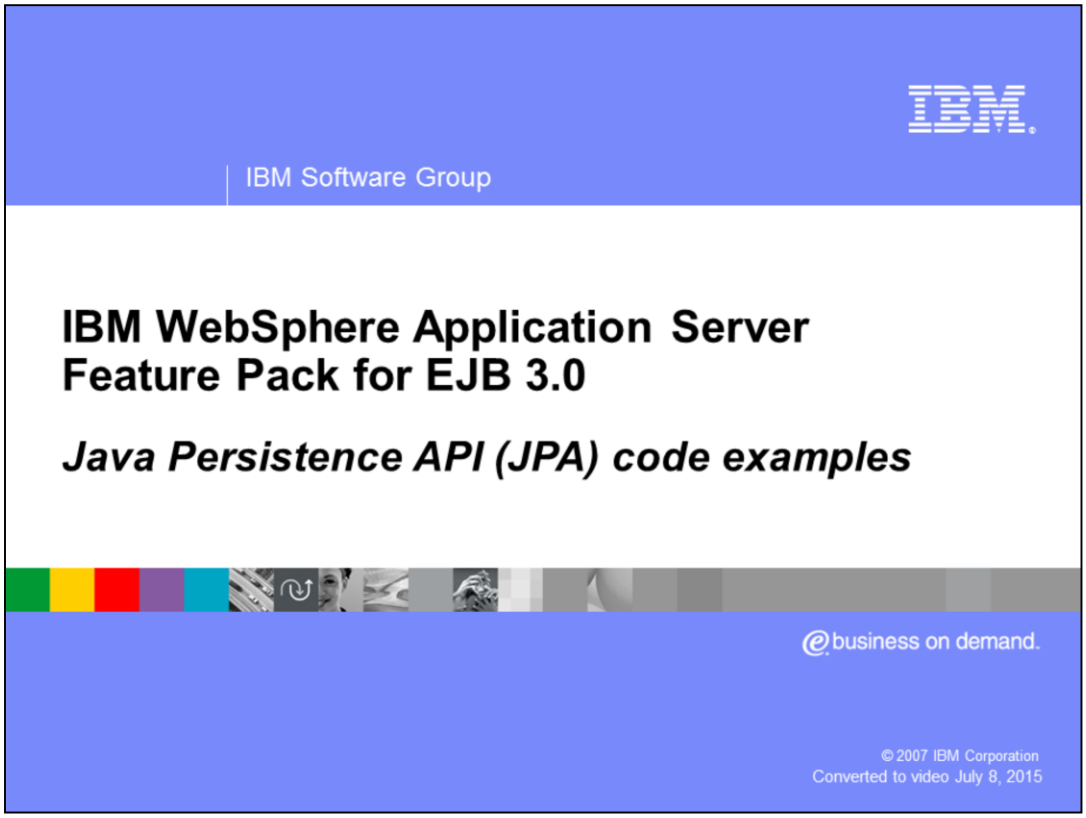
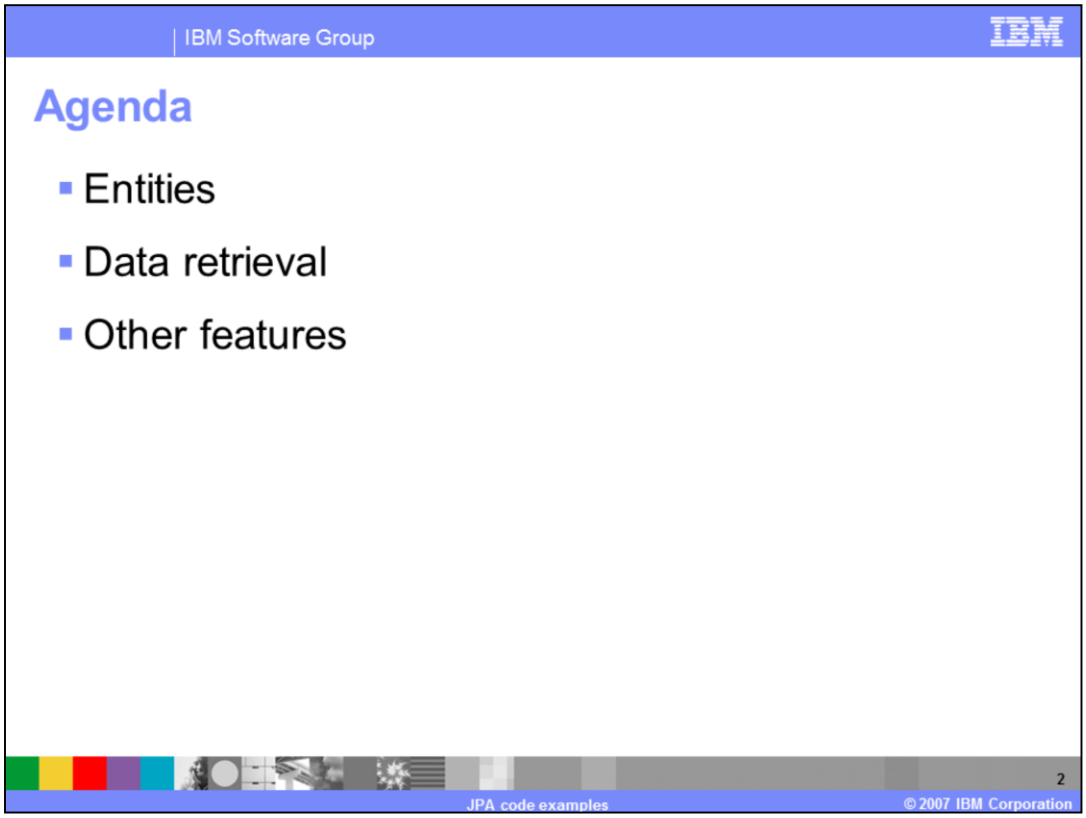
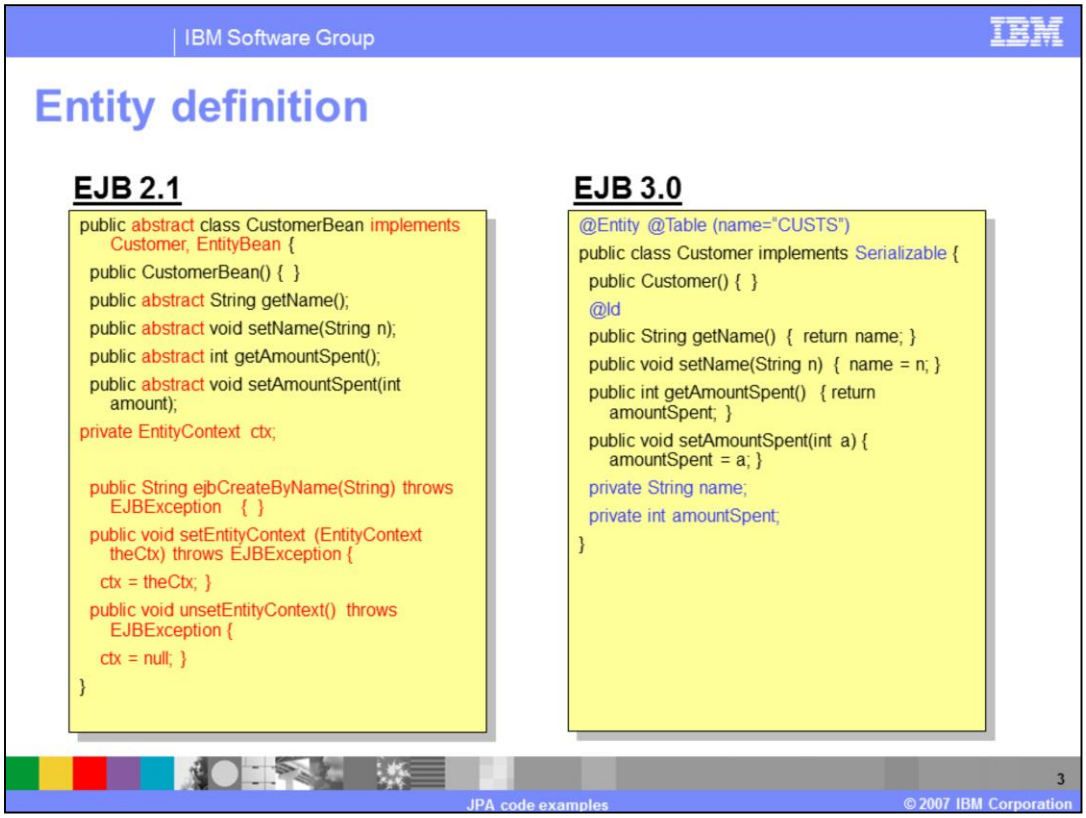


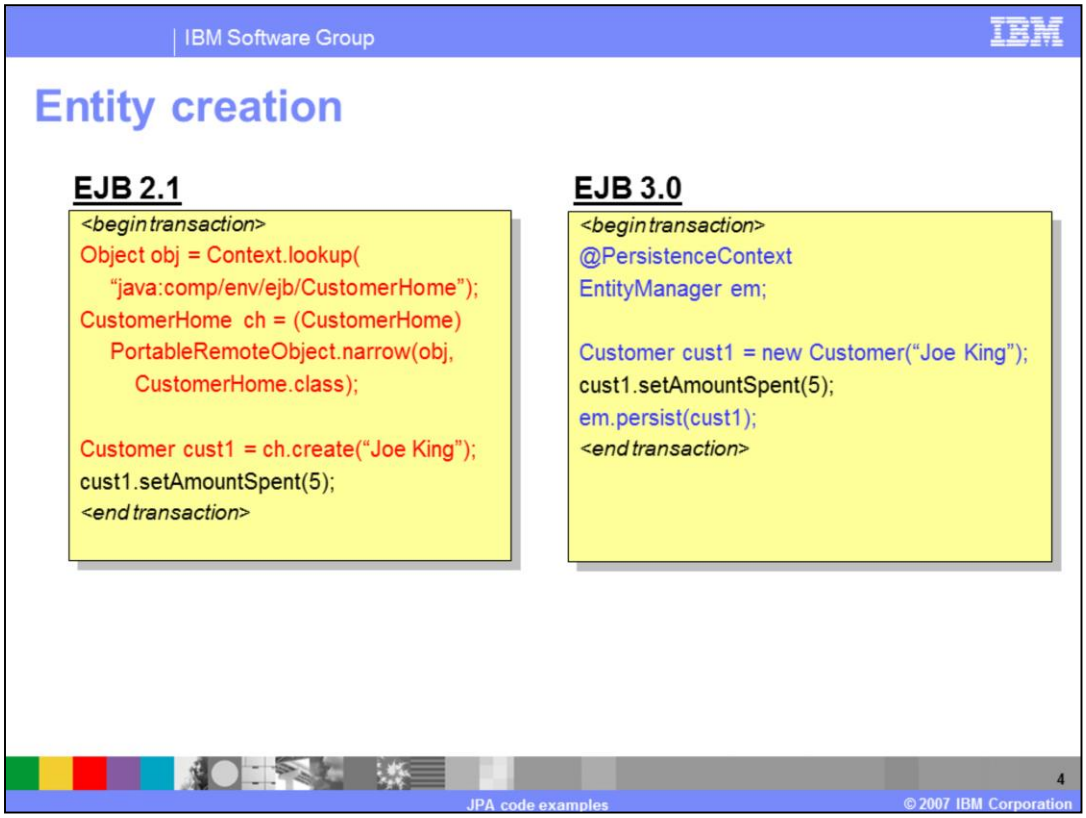
This presentation will introduce the basic concepts of the Java™ Persistence API by presenting several code examples.



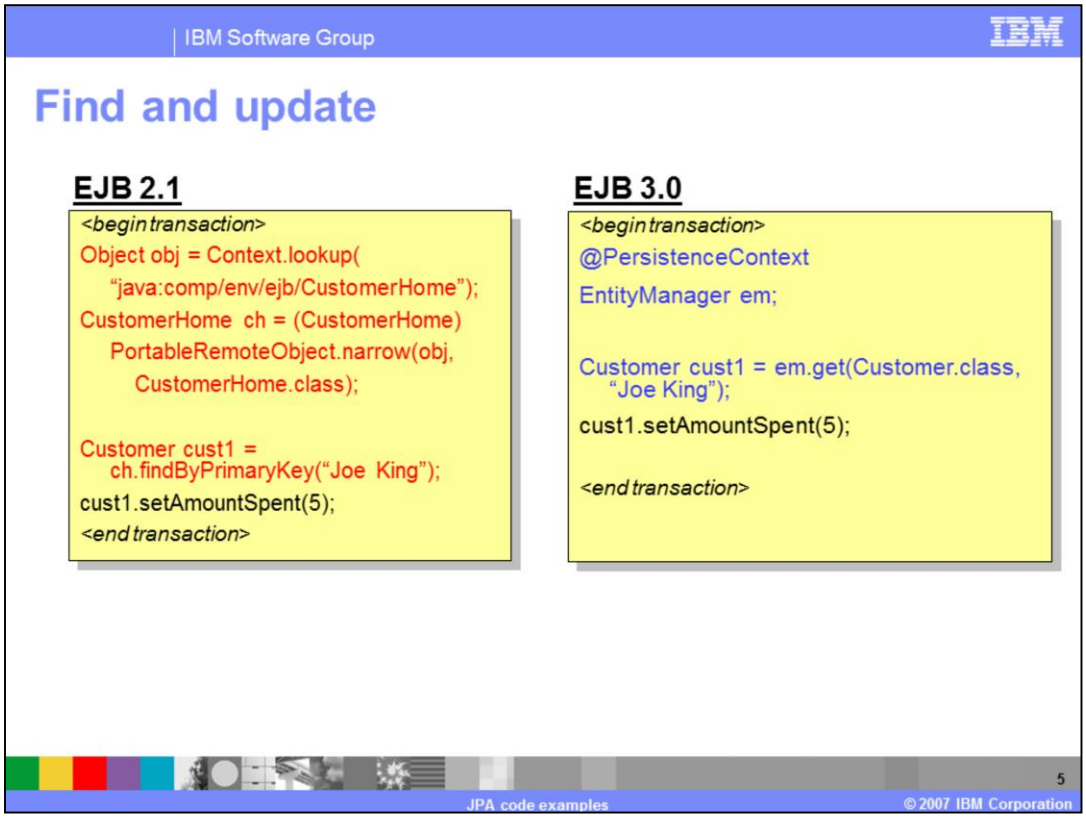
This presentation will begin with examples of how to create and use JPA entities. Data retrieval and queries will be discussed next, followed by some more advanced features of JPA.



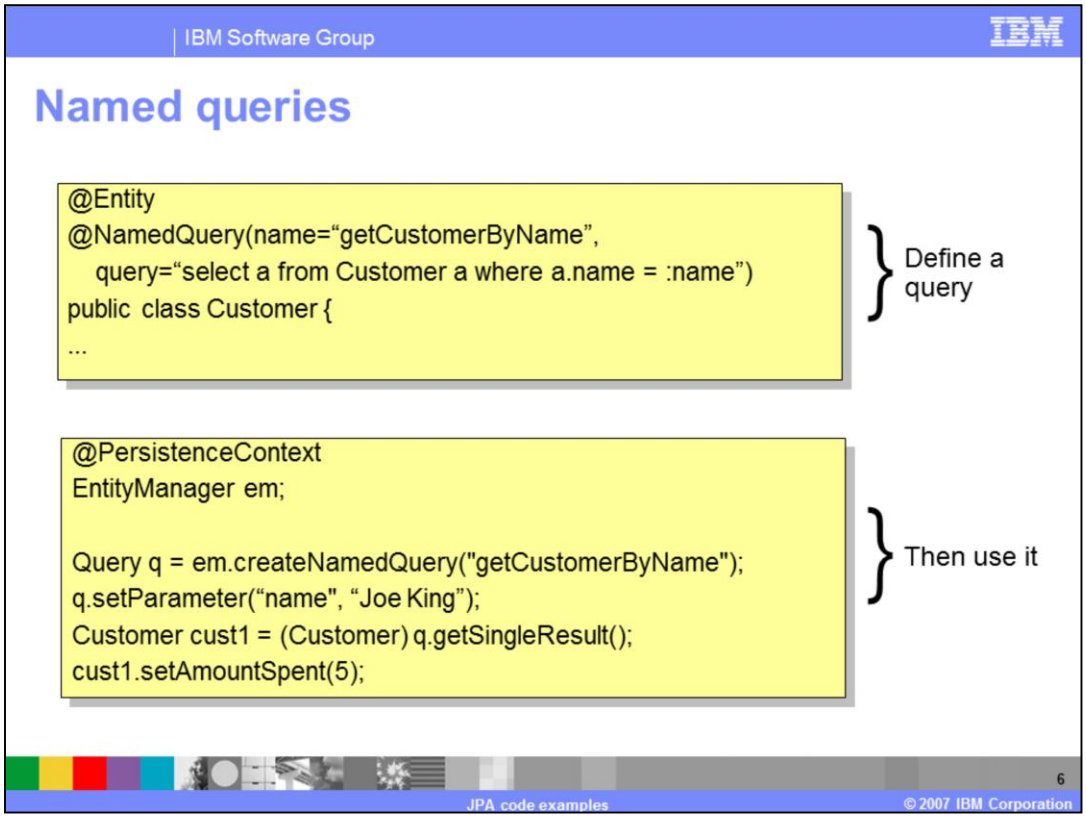
This first example compares the definition of an EJB 2.1 entity bean with a JPA Entity in EJB 3.0. The most immediately apparent difference is that unlike an EJB 2.1 entity bean, a JPA entity is a concrete, not abstract, class. You will also notice that the JPA entity does not need to include the boilerplate code, shown in red, that is included at the bottom of the entity bean example. The JPA entity is a plain Java class, with some annotations added to provide metadata and identify the class as an entity. The “@Entity” annotation indicates that the class is a JPA entity. The “@Table” annotation is optional, and specifies the name of the database table that should be used to store instances of this entity. If a table name is not specified, it is implied that the table has the same name as the class. A column in that table will be used for each member variable, and the “@Id” annotation specifies which column should be used as the primary key. Beyond the simplified code in the JPA example, it does not require any additional information to be specified in an XML deployment descriptor.



Entities are created using the “new” command, just like any other Java object. You do not have to deal with the container and perform a lookup, like you are required to do with EJB 2.1. The “@PersistenceContext” annotation is used to inject an instance of an entity manager, which you can then use to interact with the database. In this example, a new customer object is created, and then added to the database using the “persist()” method. For qualities of service, like transaction support or security, you should use the façade model and wrap the entity with a session bean.



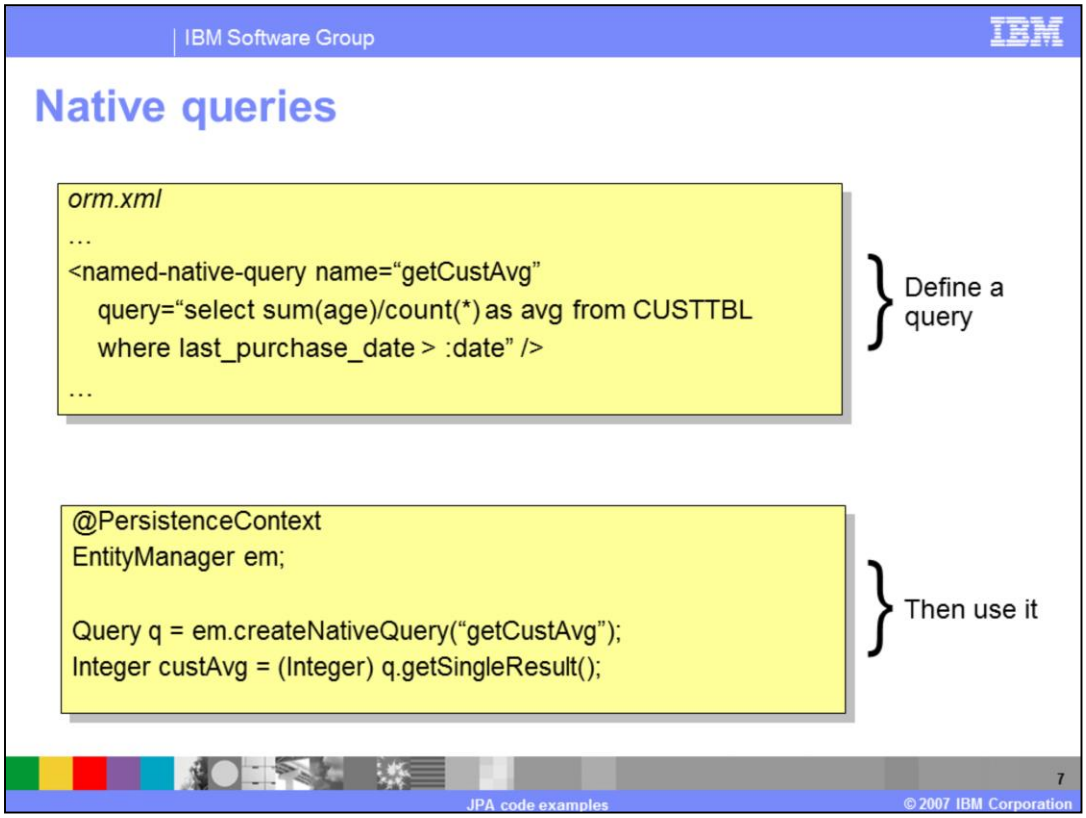
This example shows a connected update, using the entity manager's "get()" method to find and retrieve the Customer object using its primary key. Again, it is much simpler to do this with the JPA entity than with the EJB 2.1 entity bean. Disconnected updates are also available using the merge() operation to optimistically update entities that have been detached.



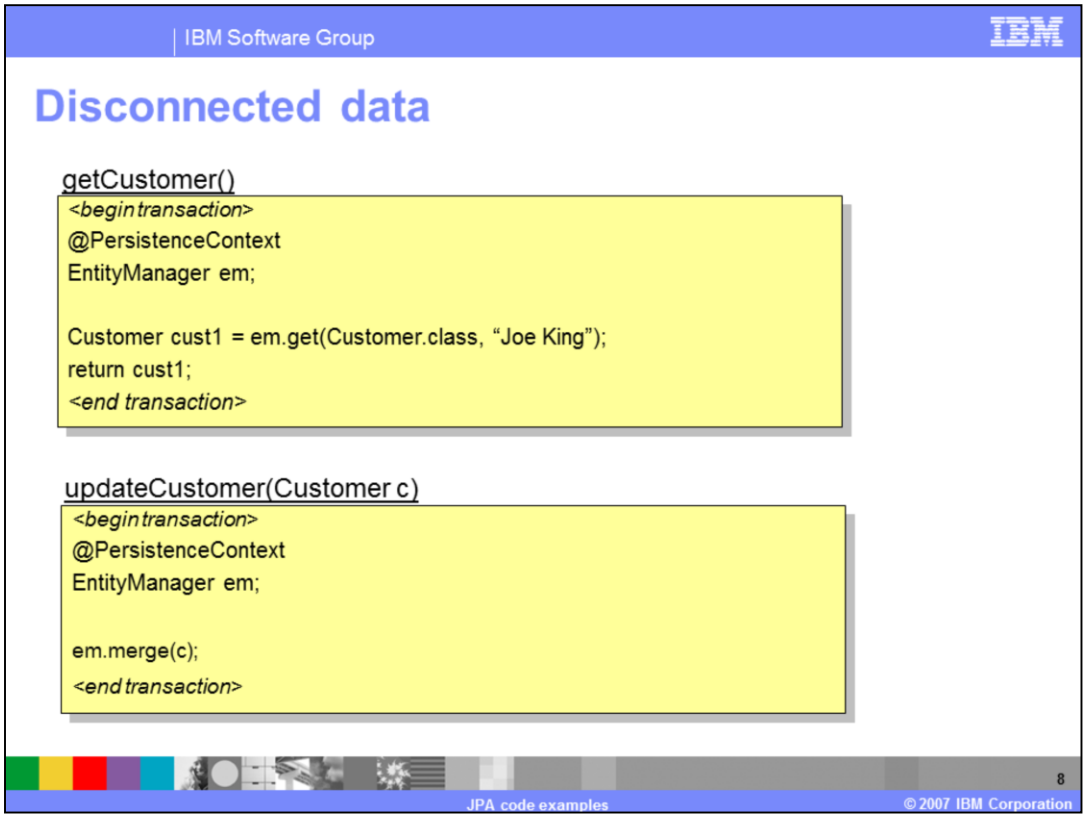
In this example, a JPQL (Java Persistence Query Language) query is defined with one variable, "name". In the lower code block, the "name" parameter is set, and then a customer object is retrieved using the getSingleResult() method on the query. The Customer object can then be acted on just like the Customer object that was found in the previous example.

In EJB 2.1, you would use EJB-QL to define the query in the deployment descriptor, and then use the bean's home to call the query that you defined in the deployment descriptor to return a Customer object.

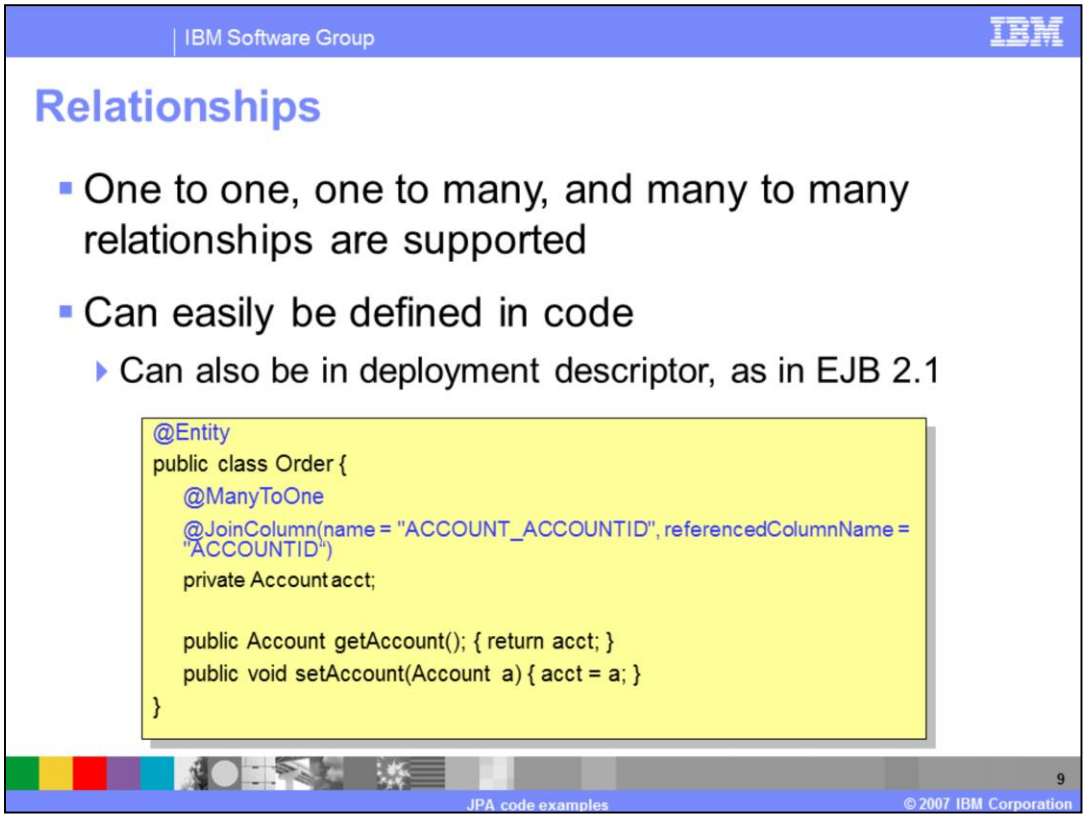
JPQL supports both named parameters and ordered parameters, whereas EJB-QL only supports ordered parameters. In addition to defining the query with an annotation, as shown here, a JPQL query can also be defined in an XML file. Defining queries in XML can be a time saver, so that you do not have to recompile your application to modify a database query.



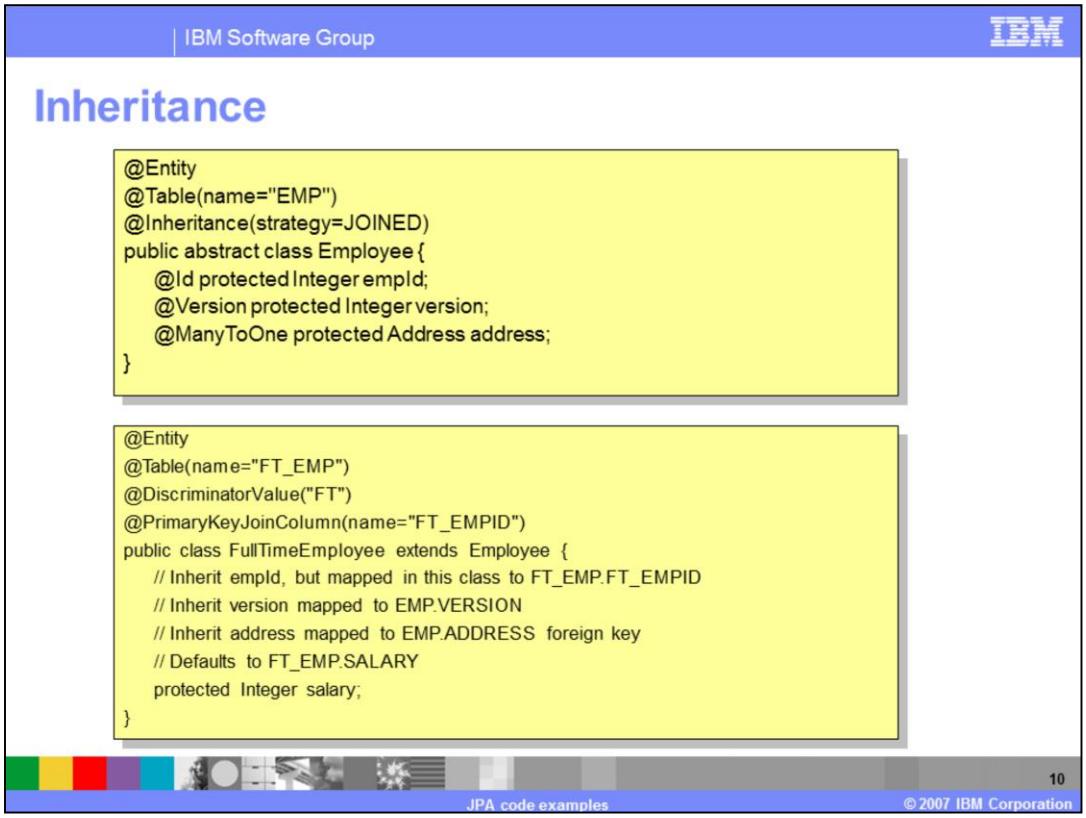
This example shows the definition of a native query in an XML file, and the invocation of that query from Java code. Calling a native query is very similar to using a JPQL query, as was shown previously. Like other query types, native queries can be defined in either Java code or XML files. It may be advantageous to define native queries in XML files so that your application code is not tied to any particular underlying database product. This approach also enables you to fine tune the SQL in your query without having to recompile your Java code.



Another feature of JPA is that an object can be returned from the database, and then be manipulated outside the scope of a transaction. When the modified object is to be committed to the database, use the entity manager's merge() method to optimistically update the database. The entity, in this case a Customer object, must define a version control field for optimistic updating. The version control field is denoted using the @Version annotation on a variable.

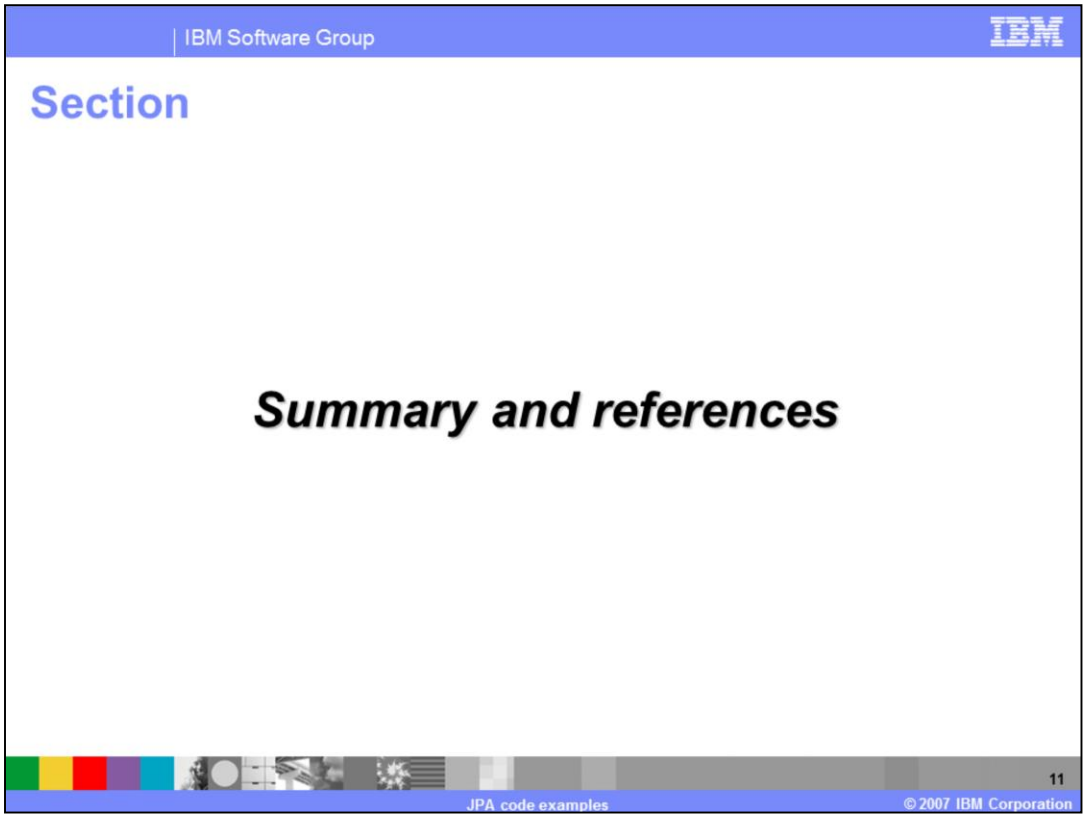


JPA supports several types of relationships, including bidirectional ones. Relationships can be one-to-one, one-to-many, or many-to-many. You can use a deployment descriptor to define relationships, as was required with EJB 2.1, or you can use Java annotations, as is shown in this example. This example shows a many-to-one relationship between Order and Account entities. The “@JoinColumn” annotation specifies the mapping between database columns in the different tables. Cascading for related entities can be specified for various operations, such as deleting, and lazy semantics are supported.

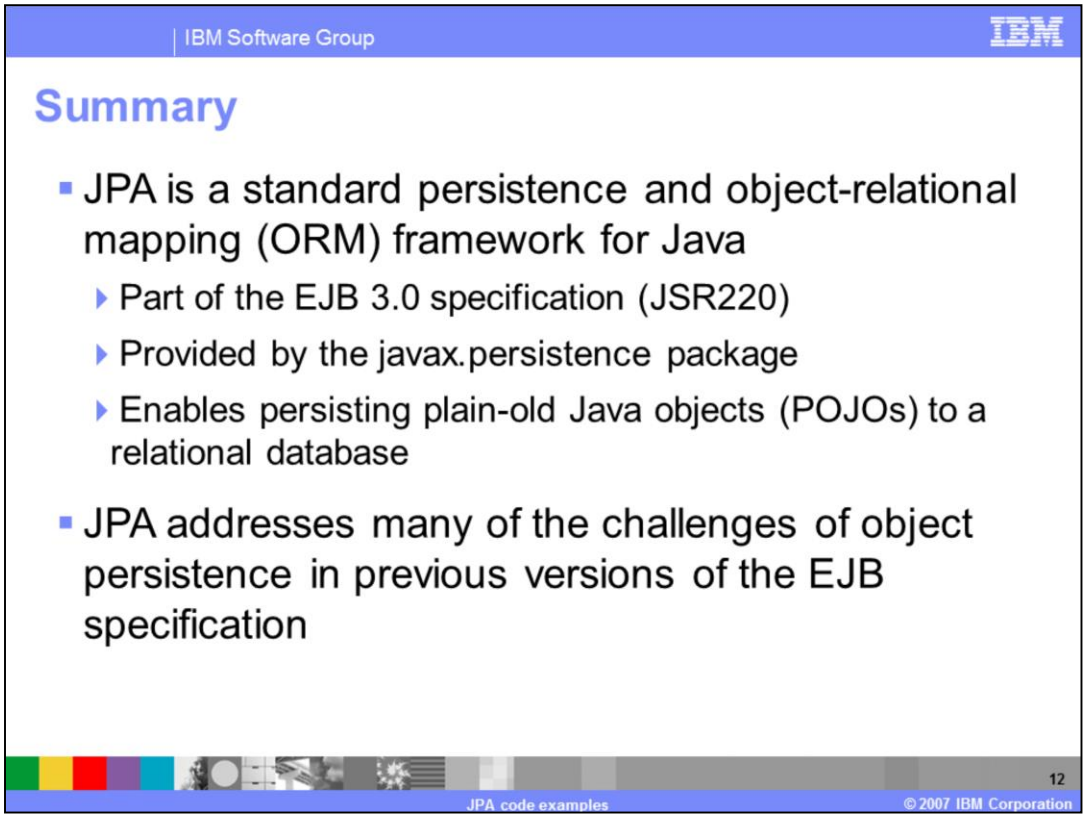


Inheritance is also supported for JPA entities. This example shows a relationship between a base class called “employee” in the first block of code, and the second block of code shows the “FullTimeEmployee” class.

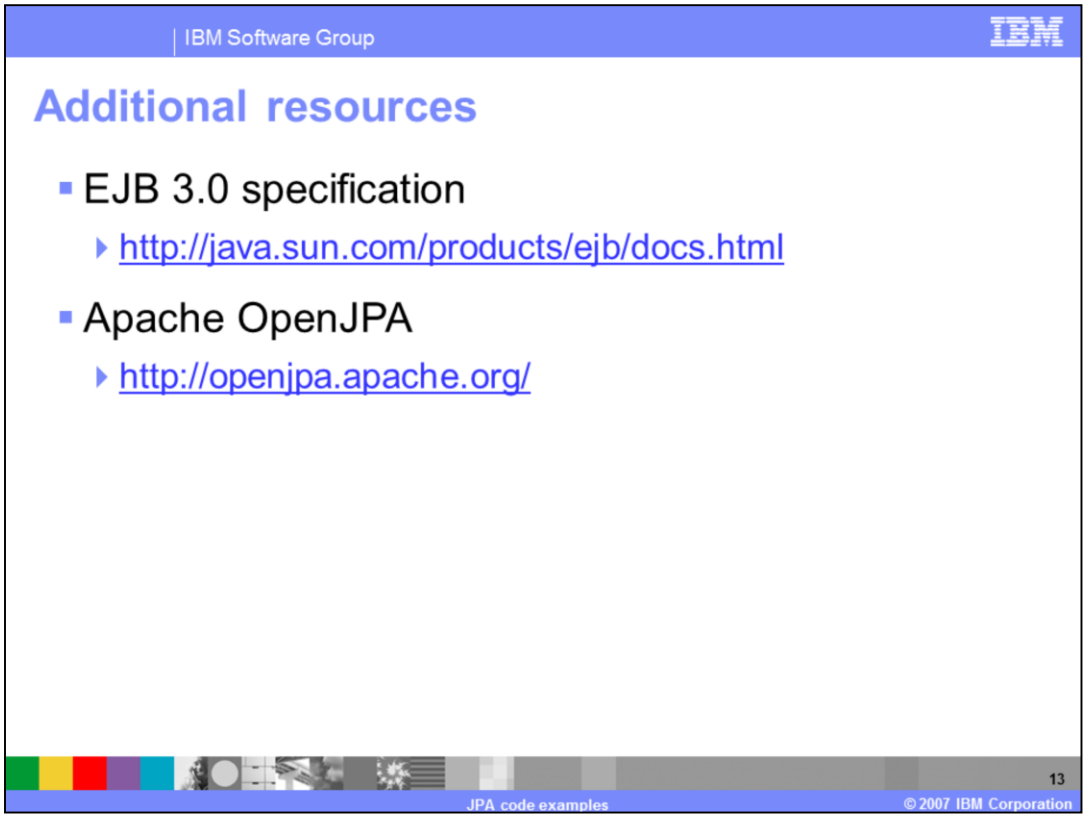
The base class, employee, is mapped to a database table named “EMP” and defines a few basic variables including a primary key and a version identifier. The FullTimeEmployee class extends Employee, adding a salary variable. FullTimeEmployee objects are mapped to the “FT_EMP” table. The empId primary key is inherited from Employee, but for FullTimeEmployees it is stored in a column called “FT_EMPID”, as is specified with the “@PrimaryKeyJoinColumn” annotation. As with other JPA features, inheritance can also be specified using XML deployment descriptors.



This section will summarize the presentation.



The Java Persistence API is a standard persistence and object-relational mapping framework that is part of the EJB 3.0 specification. It uses plain-old Java objects to represent items in a relational database. It is designed to be a persistence framework for Java EE that improves on prior EJB persistence models by addressing their inherent challenges, by being less complex and lighter-weight. The code examples in this presentation have shown how to create and use entities, how to query entity data using the entity manager, and how to define relationships and inheritance between entity classes.



This slide lists some resources that may be helpful for learning more about JPA. The JPA specification is a sub-set of the EJB 3.0 specification.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

EJB, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.