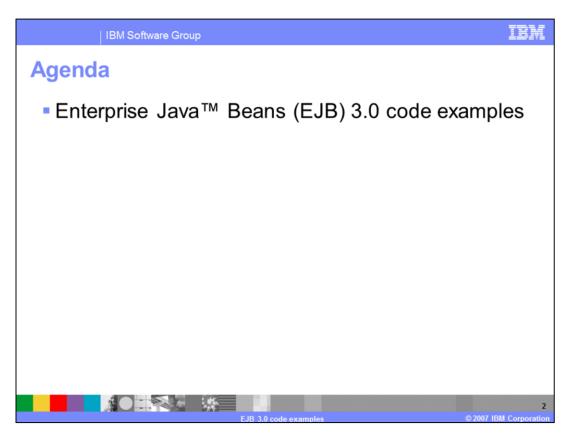
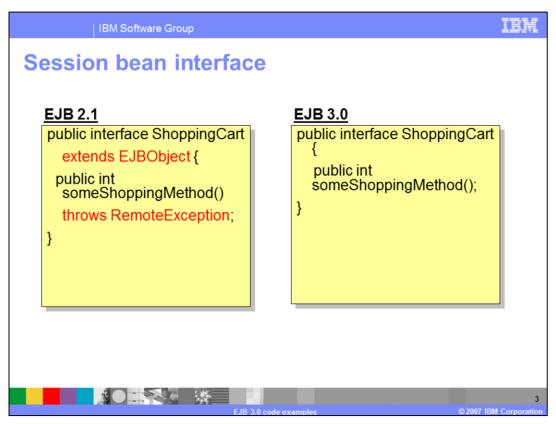


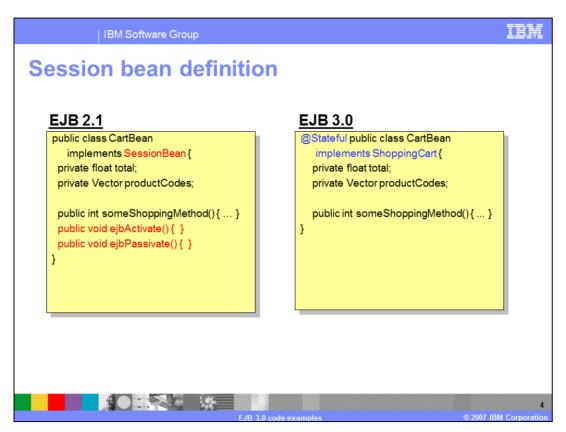
This presentation will cover basic EJB 3.0 code examples.



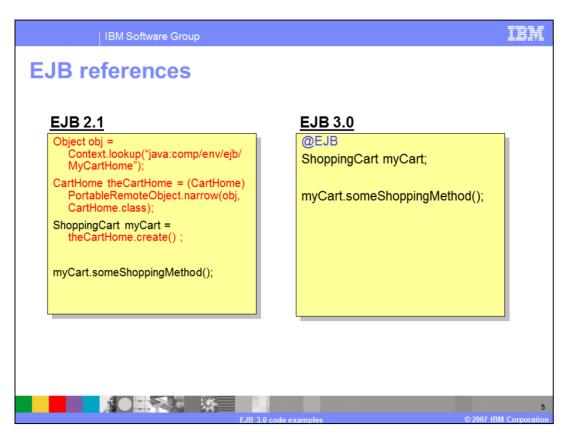
In this presentation, you will see several basic code examples. Many of them will show EJB 2.1 code alongside equivalent EJB 3.0 code, to help you understand the differences between the two programming models.



This first example shows a simple session bean interface. In EJB 2.1, your interface must extend EJBObject, and you are required to throw a RemoteException. The EJB 3.0 interface is just a plain Java interface, with no special requirements. In EJB 3.0, bean interfaces are optional. If you do not create one, it will be generated automatically at runtime. One reason that you might want to create your own interface is if you want to selectively expose the public methods on your bean. When an interface is automatically generated, all public methods are exposed.



In this example, note that the EJB 3.0 session bean is a plain-old Java object (POJO) that implements its own interface, not "SessionBean". It is recognized as a stateful session bean because of the "@Session" annotation that precedes the class definition. EJB 3.0 also does not require implementation of the callback methods you seen in the EJB 2.1 beans, ejbActivate() and ejbPassivate(). You need only implement them if you intend to use them. If you did not create a business interface, all of your public methods will be exposed by the automatically-generated interface. Alternately, you can use the "@BusinessMethod" annotation to selectively expose only certain methods. Note that in this basic example, the EJB 2.1 code would also require an XML deployment descriptor, while the EJB 3.0 code can stand on its own and run without any additional metadata.



Here you see the difference between EJB 2.1 and 3.0 when it comes to EJB references. With EJB 2.1, if you want to call an EJB from the Web container or from another EJB, you need to first lookup the bean's home interface, then use "narrow()" to cast the object to the appropriate interface type, and call "create()" to instantiate the bean. With EJB 3.0, you can use the @EJB annotation to easily inject an instance of an EJB, as shown in the example on the right. The example injects an instance of ShoppingCart named myCart, and you can then call methods on that object as you normally would.

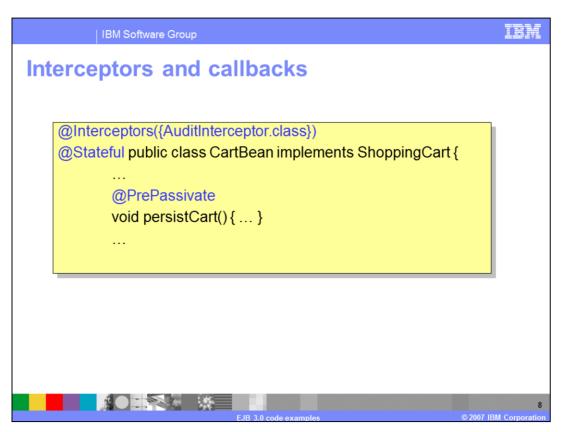


This slide shows a basic EJB 3.0 message-driven bean, designated as such by the "@MessageDriven" annotation. You can use the "@ActivationConfigProperty" annotation to provide information for the bean to use, such as destination and connection factory JNDI names. You must also provide a bindings file so that the bean will be able to locate the objects that exist with these names. Bindings files are discussed in detail in the Information Center. It is not necessary to inject the MessageDrivenContext in this example, but it is shown here to demonstrate how it can be injected to give you access to information maintained by the container, such as transaction information.

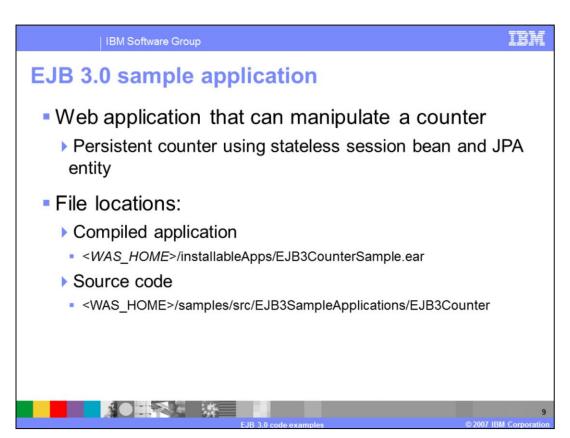
```
Interceptors

public class AuditInterceptor {
          @AroundInvoke
          public Object logMethodInvocation(InvocationContext inv)
          throws Exception {
                Log.log("Method "+inv.getMethod()+" invoked.");
                return inv.proceed();
          }
}
```

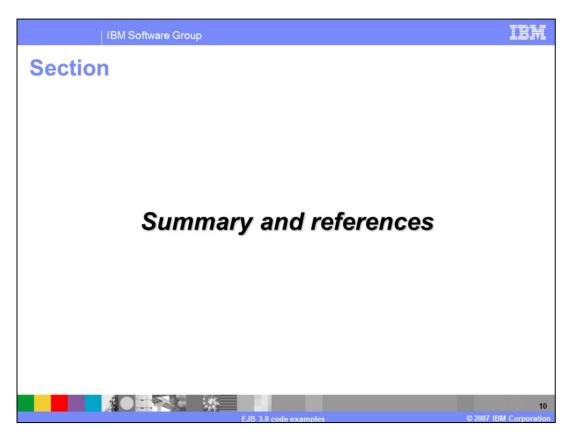
Interceptors give you the ability to do pre- and post-processing, similar to what you can do with a servlet. An interceptor can be defined within the class or in a separate class, and must take an InvocationContext as an argument. A complete list of available interceptors can be found in the EJB 3.0 specification. This example uses a separate class to define an interceptor that logs the name of each method being called.



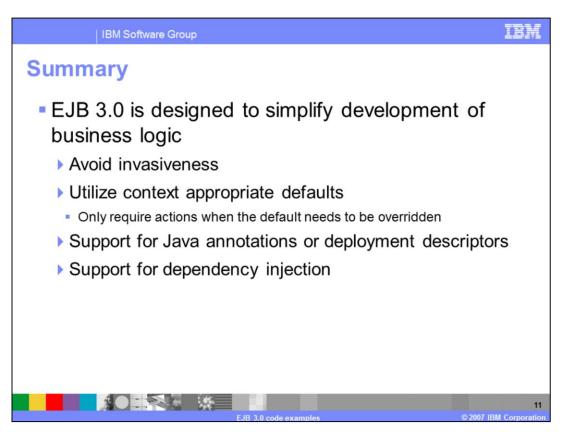
This snippet shows an example of how you specify that a class should use an interceptor. The "@Interceptors" annotation can take a list of interceptor classes as arguments. There is also an example here of how to use callback annotations. In this case, the "@PrePassivate" annotation says that the "persistCart()" method should be called immediately before the bean is passivated.



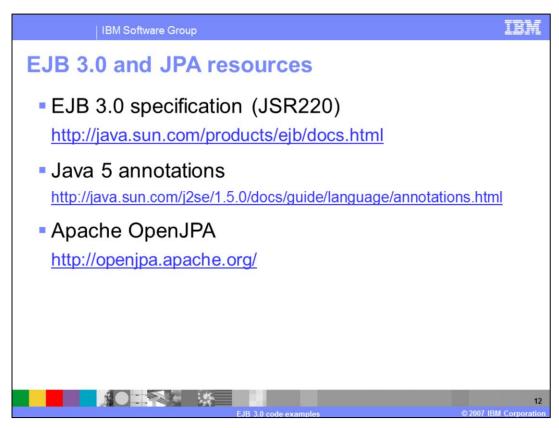
If you have installed the Feature Pack for EJB 3.0, you can find a sample EJB 3.0 application in the "installableApps" directory. The source code for the application can be found in the "samples" directory. The sample application comes with an Ant build.xml file to show how EJB 3.0 applications can be built using Ant. You can also download a lab exercise that will walk you through the process of configuring the Application Server Toolkit to support development of EJB 3.0 applications using this sample.



This section will summarize the presentation.



The EJB 3.0 specification is designed to make it faster and easier to develop your business logic. By introducing Java annotations for EJBs, dependency injection support, removing boilerplate code, and making use of intelligent defaults, EJB 3.0 provides a framework that enables anyone with Java skills to develop enterprise beans. The code snippets in this presentation have given basic examples of how create interfaces, session beans, and message-driven beans, and also how to use interceptors and callback methods. The samples also show how the POJO-based EJB 3.0 framework is simpler than EJB 2.1.



This slide lists some resources where you can learn more about EJB 3.0 and JPA. You should also consult the Feature Pack for EJB 3.0 Information Center for more information.

## 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:

M 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.

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 MPLIED. IBM EXPRESSLY DISCLAUMS ANY WARRANTY SOFT MERCHANTABLITY, FITNESS FOR A PARTICULAR PURPOSE OR NONHERINGEMENT. IBM shall have no responsibility to update his information. Bit products are warranted, if at al., according to the terms and conditions of the agreements (for example, IBM Customer Agreement, National Program Licenses Agreement, let under whorted), there mitional Program Licenses Agreement, let under whorted they are provided information concerning non-IBM products was obtained from the suppliers 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.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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.

© 2007 IBM Corporation