

Enterprise Application Development in Java EE

Enterprise Application Development in Java EE Trainer's Guide

© 2015 Aptech Limited

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2015



Dear Learner,

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner.

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as learning-to-learn, thinking, adaptability, problem solving, positive attitude etc. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence, considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

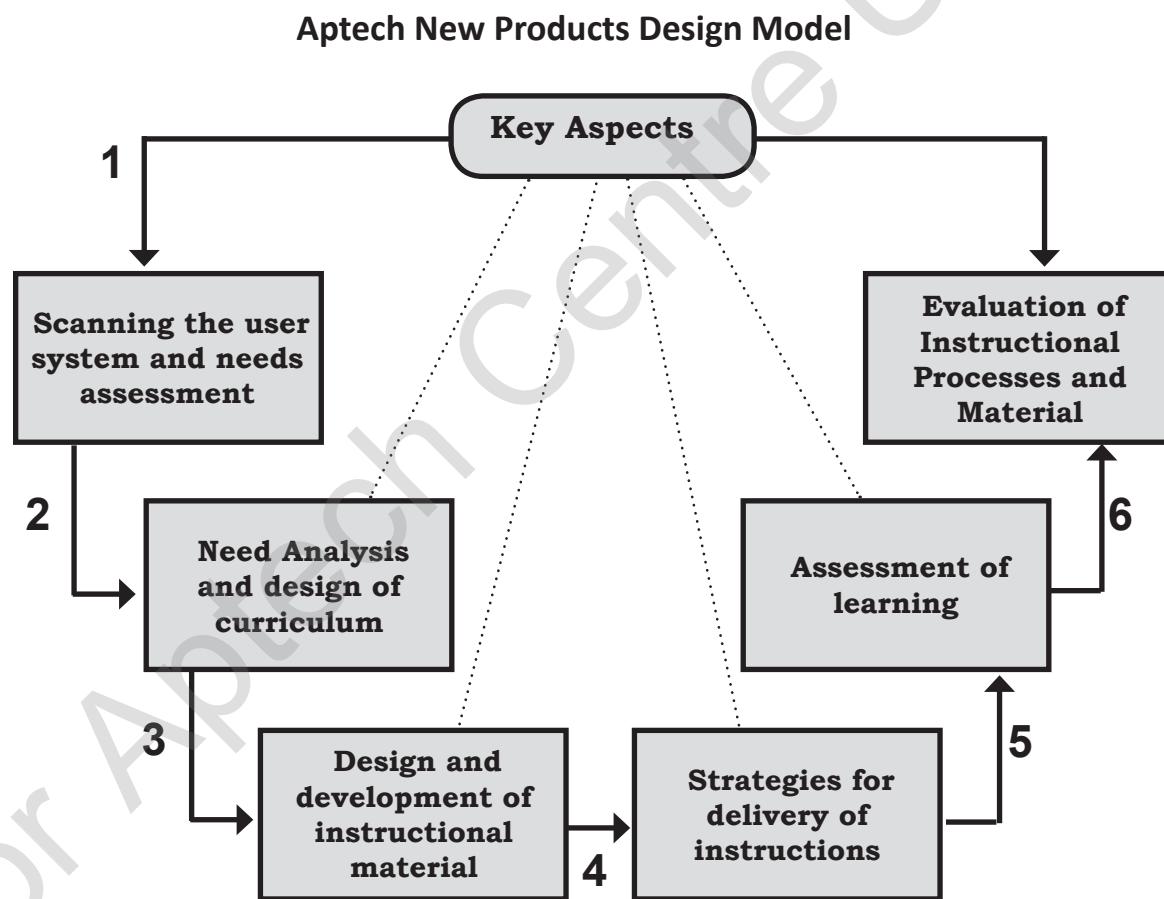
The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

- Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.



“

A little learning is a dangerous thing,
but a lot of ignorance is just as bad.

”

Preface

The book 'Enterprise Application Development in Java EE' Trainer's Guide aims to teach the students how to design and develop enterprise applications on Java Enterprise Edition (Java EE) platform. The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the module and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

“ Practice is the best of
all instructors.



Table of Contents

Sessions

1. Introduction to Business Components
2. Enterprise JavaBeans
3. Session Beans
4. Stateful Session Beans
5. Singleton Session Beans
6. Introduction to Messaging
7. Interceptors and Dependency Injection
8. Transactions
9. Persistence of Entities
10. Advanced Persistence Concepts
11. Query and Criteria API
12. Concurrency, Listeners, and Caching
13. Security
14. EJB Timer Service
15. EJB Design Patterns

**“ The future depends on what
we do in the present.**

Session 1 – Introduction to Business Components

1.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two which will be a key point to relate the current session objectives.

1.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain enterprise applications
- Describe the problem faced by large enterprise applications
- Explain distributed application architecture used for developing business components
- Describe Java EE platform and its API
- Describe application development architecture of enterprise application
- Describe application server containers and their services
- List various Java EE application servers
- Describe Java EE profiles and the use of EJBLite on application servers

1.1.2 Teaching Skills

To teach this session successfully, you should be aware of what enterprise applications are and the process for developing them using Java EE platform. You should also be aware about distributed application architecture used for building enterprise applications. You should have sound knowledge about component based model of application development as Java EE uses these concepts to build enterprise applications.

The session also covers an introduction to Application Programming Interfaces (APIs) provided by Java EE platform for enterprise application development. You should aware yourself with different Java application servers used for deploying and executing enterprise applications. You should also gain knowledge about services provided by different types of containers provided by Java application servers. Finally, update yourself with the features of Java EE 7 platform and its new API. Also, aware yourself with the EJBLite and the profiles on Java EE 7 platform.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- Explain enterprise applications
- Describe the problem faced by large enterprise applications
- Explain distributed application architecture used for developing business components
- Describe Java EE platform and its API
- Describe application development architecture of enterprise application
- Describe application server containers and their services
- List various Java EE application servers
- Describe Java EE profiles and the use of EJBLite on application servers

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 2

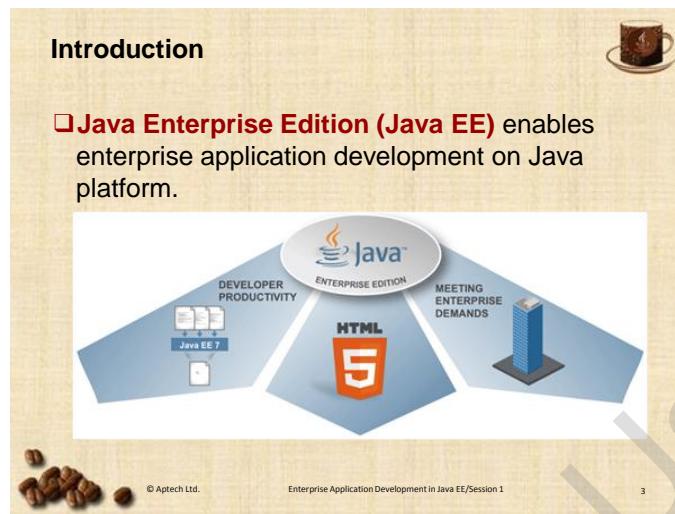
Tell them that they will be introduced to enterprise application development using Java EE in this session. Java EE stands for Java Enterprise Edition which is a platform for developing enterprise applications. Tell them about the enterprise edition of the platform and various APIs provided by it.

Also, tell them that Java EE provides different APIs for transaction management, accessing the database and providing security. These are essential components of the enterprise application; Java EE provides these services through the container. Tell them about the utility of application servers in enterprise applications. Finally, the session discusses about Java EE profiles and EJBLite.

1.2 In-Class Explanations

Slide 3

Let us understand the composition of Java EE platform.



Use slide 3 to explain the composition of Java EE platform.

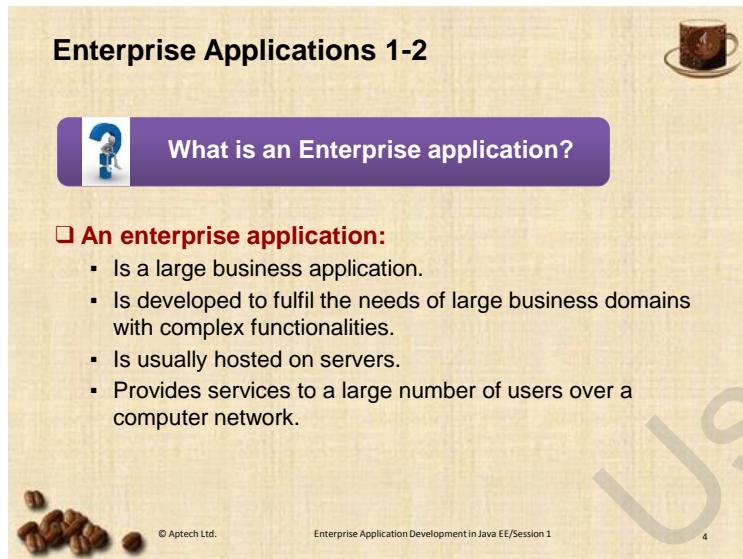
Java has come a long way from its humble beginning as a platform-neutral programming language for consumer electronic devices. Today, it has grown in various directions with the objective of fulfilling the requirements of various industries and domains.

The different flavors of Java are differentiated based on the packages provided for the development of various kinds of applications. One of the most interesting packages provided by Java is Java Enterprise Edition (Java EE) platform that enables building of enterprise applications. Some of the features offered by Java EE 7 platform are as follows:

- **Java EE 7 can improve developer productivity** – Java uses component-based application development where each component is a reusable entity. An application is built with various components working together. Being reusable, these components reduce the developer effort and testing effort of the developers. Thus, considerably reducing the application development time.
- **Java EE 7 is useful for Web-based applications** – Java EE 7 is compatible with HTML5 which is widely used in Web-based enterprise applications. All the Web components of the application are deployed in a Web container, where the Web container provides all the generic services required for Web connectivity.
- **Java EE 7 enables applications to meet enterprise demands** - An enterprise application should be able to cater to changing number of end users without deteriorating the application performance. Java EE 7 provides mechanisms to improve the scalability of the enterprise application, as the number of end users using the application increases.

Slide 4

Let us understand about enterprise applications.



The slide has a title 'Enterprise Applications 1-2' at the top left. On the right side, there is a small icon of a coffee cup with steam. At the bottom left, there is a small illustration of coffee beans. The main content area has a purple header bar with a question mark icon and the text 'What is an Enterprise application?'. Below this, a red square contains the text '□ An enterprise application:' followed by a bulleted list of four points. The footer of the slide includes the text '© Aptech Ltd.', 'Enterprise Application Development in Java EE/Session 1', and the number '4'.

- Is a large business application.
- Is developed to fulfil the needs of large business domains with complex functionalities.
- Is usually hosted on servers.
- Provides services to a large number of users over a computer network.

Use slide 4 to explain about an enterprise application.

An enterprise application is a large business application with distinct features of the application domain. Each enterprise application is designed and developed to fulfil the requirements of the context for which it is developed.

For instance, consider the case of railway reservation system. The application has to allow users to reserve tickets for different trains of the railway service, allow the end user to cancel tickets reserved by them, and modify the reservations made by them. Apart, from the basic functionality of managing tickets, the application is expected to support varying number of users who are simultaneously accessing the application. The railway reservation service is expected to be available to multiple users at the same time with minimum failures.

In-Class Question:

After you finish explaining the enterprise applications, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



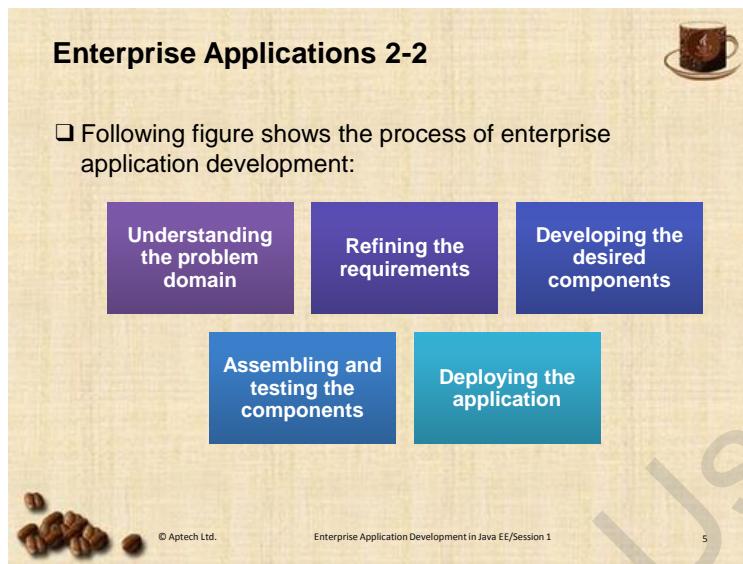
Can you name some other application domains that require enterprise applications?

Answer:

Banking domain, financial domain, employee portals in companies, and so on.

Slide 5

Let us understand the enterprise application development process.



Use slide 5 to explain different stages in the enterprise application development process.

Before actually writing code for the application, there are certain design tasks to be carried out for the enterprise application.

The application design process includes understanding the application domain. This phase is also known as requirements analysis. The requirements and constraints with respect to the application are collected and documented during this process. The documents generated in this phase are usually text documents.

After collecting the requirements, they are put into structured documents such as use case diagrams, class diagrams, and so on. These enable development of the object-oriented application. Refining the requirements involve the steps such as turning the user requirement into a form which can be used by the developer to build the application.

With the refined requirements which are developed as the outcome of collecting the requirements, developers create the application components. Developers create classes, respective methods and define the interaction of objects in each application component.

The components of the application are independently developed. These components have to be integrated together to work as a single enterprise application. Components are integrated and tested.

Once the application is tested and it is ensured that the application is functioning as expected, the application is deployed on the application infrastructure.

Slides 6 and 7

Let us understand the need for enterprise applications.

Need for Enterprise Applications 1-3



Current Scenario:

- The rapid changes have made businesses to meet the requirement of customers, communicate with other business processes, and incorporate business-to-business services.
- Enterprise applications are developed to satisfy such business needs.



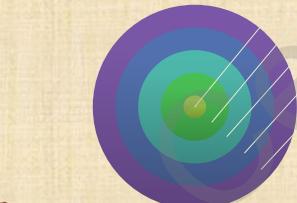
© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 6



Need for Enterprise Applications 2-3



Following figure shows some different types of domains involving complex business functionalities:



Banking
Logistics
Finance
Production Scheduling
Order Management



© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 7

Use slides 6 and 7 to explain the needs of an enterprise application.

Using slide 6 tell them that the enterprise applications are required to fulfill the requirements of large businesses and to efficiently operate businesses. They are used in businesses which span over multiple geographical locals and are used by large number of users. The business processes in these applications may also interact with other business processes in the domain. Enterprise applications are developed to satisfy the needs of such businesses. Then, using slide 7, explain that an enterprise applications are developed for the following domains – Banking, Logistics, Finance, Production scheduling, Order management, and so on.

Tips:

Apart from Java EE platform enterprise applications can also be developed through .NET platform. Provide an introduction to .NET platform as added information to the students. You can use the following link for this purpose: [https://msdn.microsoft.com/en-us/library/aa244216\(v=vs.60\).aspx](https://msdn.microsoft.com/en-us/library/aa244216(v=vs.60).aspx)

Slide 8

Let us understand the concerns and requirements of the large businesses.

Need for Enterprise Applications 3-3

Concerns and requirements of large businesses:

- Should be long lived applications performing parallel processing.
- Should be functional across platforms.
- Should support complex business processes and domain-based constraints.
- Should follow procedures with respect to security, administration, and maintenance of the applications.
- Should follow complex business requirements that are defined through policies, constraints, rules, processes, and entities in the domain.

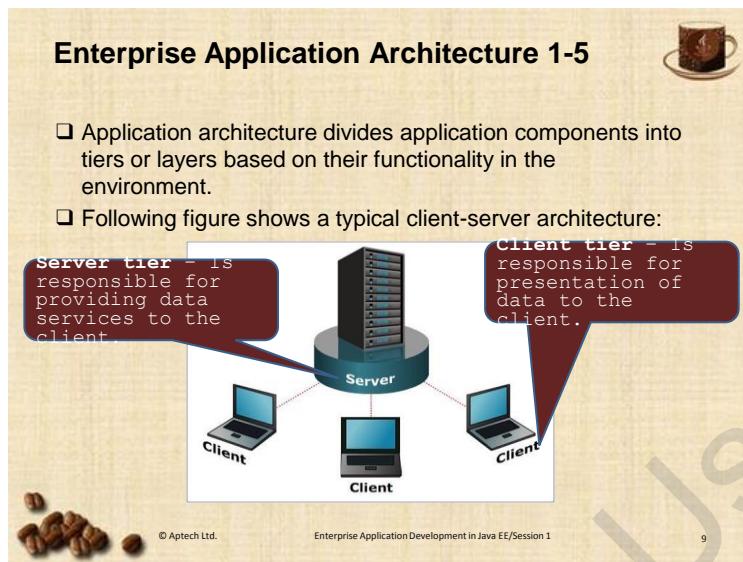
© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 8

Use slide 8 to explain various requirements and concerns of large businesses.

- a. Large businesses require applications which run for long duration and serve multiple users. These applications should implement parallel processing techniques for better performance.
- b. Large applications are used by a wide range of users with different hardware and software configurations, the application should work appropriately on these hardware and software platforms.
- c. The enterprise application is expected to model all the complex business processes of the application domain and also hold the domain constraints. Here, you can use an example of an application such as banking application and describe what are the business processes in the banking application along with the domain constraints in it.
- d. The enterprise application data is vulnerable to malicious attacks; hence the application should implement security mechanisms. The application should have proper system administration and application maintenance mechanisms in place.
- e. All the requirements, constraints, and security policies of the application should be properly implemented.

Slide 9

Let us understand the application architecture such as client-server.



Use slide 9 to explain the application architecture.

Application architecture divides application components into tiers or layers based on their functionality in the environment.

Tell them that the earlier applications were designed on two-tier architecture. A two-tier architecture is a software architecture in which a presentation layer or interface runs on a client, and a data layer or data structure that gets stored on a server. Separating these two components into different locations represents two-tier architecture, as opposed to a single-tier architecture.

In client-server architecture, the server hosts all the application services. The enterprise application is deployed on the server. Whenever a client requires a service it has to connect to the server and place a request for the service. The server responds to this request and provides the service. For example, Web applications such as mail applications are also based on client-server architecture, mails of all the end users reach their mail boxes hosted on the mail server. Users can access their respective mail boxes by logging into their mail boxes with their username and password.

Client-server is also termed as a two-tier architecture, where the client tier represents all the application clients and the server tier represents the application server which can service the requests.

The basic client server architecture does not have any functionality on the client-side, such clients are also known as thin clients, as they do not do any processing logic residing on them.

Slide 10

Let us look at a variant of client-server architecture.

Enterprise Application Architecture 2-5

The diagram illustrates a variant of client-server architecture. A central server, labeled "Mainframe Server", is connected to a client labeled "Thick Client". The Mainframe Server contains two optional layers: "Business Logic Layer (Optional)" and "Data Access Layer". The Thick Client also contains two optional layers: "Business Logic Layer (Optional)" and "User Interface Layer". A dashed arrow points from the Mainframe Server's Business Logic Layer to the Thick Client's Business Logic Layer, indicating a flow of logic between them.

There is another variant of two tier architecture where the client is a thick client.

- A thick client contains some business logic residing on it.
- A significant part of the executable code resides on the client-tier.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 10

Use slide 10 to explain a variant of client-server architecture.

In this variant of the client-server architecture, the clients are known as thick clients. Thick clients have a part of business logic of the application residing on them.

You can give an example of the functionality which can reside on the thick client. One such example is validating the input provided from the end user. If the end user using the application has to provide input, the thick client can validate the input and in case of invalid input it can prompt the end user to provide correct input.

In-Class Question:

After you finish explaining features and functionalities of the client-server architecture, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



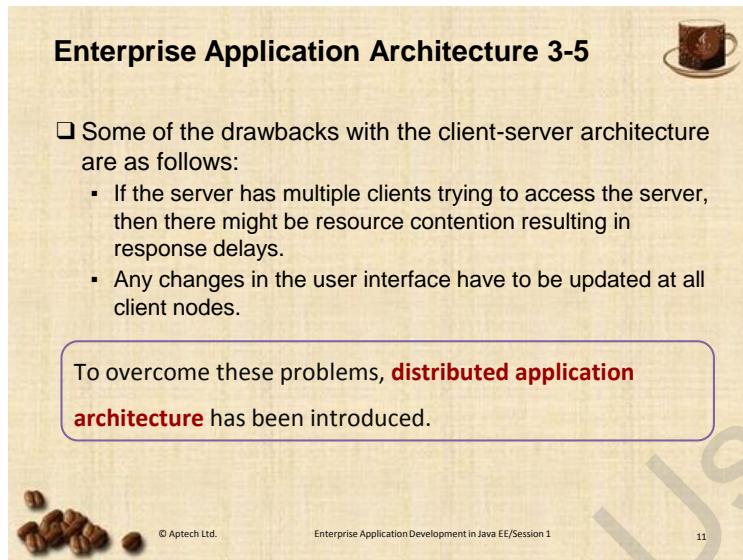
Can you give an example of a thick client and thin client for an application you have come across?

Answer:

ATM machine is a thin client as it just takes input and sends the data to be processed to a centralized server. An application registration Web page which can perform client-side validations of the data entered by the user is a thick client.

Slide 11

Let us understand the drawbacks of client-server architecture.



The slide has a parchment-like background with a coffee cup icon in the top right corner. At the bottom left are coffee beans. The title "Enterprise Application Architecture 3-5" is at the top left. A callout box contains text about distributed applications. The footer includes copyright information and a page number.

Enterprise Application Architecture 3-5

□ Some of the drawbacks with the client-server architecture are as follows:

- If the server has multiple clients trying to access the server, then there might be resource contention resulting in response delays.
- Any changes in the user interface have to be updated at all client nodes.

To overcome these problems, **distributed application architecture** has been introduced.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 11

Use slide 11 to discuss the drawbacks of client server architecture.

When all the processing is concentrated on the application server, all client requests are queued up at the server. There are instances when there are several clients indefinitely waiting for server response. Since, the processing capacity of the server is limited, the waiting time for each client request increases, and the application performance degrades. The dependency of multiple clients on a single server is the major drawback of the client-server architecture. Distributed architecture removes this dependency.

Slide 12

Let us understand distributed application architecture.

Enterprise Application Architecture 4-5

What is a Distributed application architecture?

Distributed application architecture:

- Divides the large monolithic application into layers.
- Introduces middleware layer whose main purpose is to bridge the gap between different hardware systems.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 22

Use slide 12 to explain the distributed application architecture.

To overcome the drawbacks of client-server architecture, distributed application architecture was developed. When there is a single application server, all client requests are queued up at the server. In order to reduce the waiting time, distributed architecture introduces multiple server instances of application server rather than a single server. Single server also increases the risk of application failure due to server crash; in case of multiple server distributed architecture this risk is mitigated.

Also, in case of distributed architecture one large application is divided into multiple tiers. For each tier, functionality can reside on a single machine or can span over multiple machines of the application infrastructure.

This architecture also supports multiple servers and different locations providing the same application service. When the application is distributed over multiple servers, the incoming client traffic is also distributed among the servers. This distribution of traffic in turn improves the application performance.

However, maintaining multiple servers requires additional functionality to be implemented in the application such as maintaining consistency among different application servers of the application, identifying the server to which the current application request should be routed, and so on.

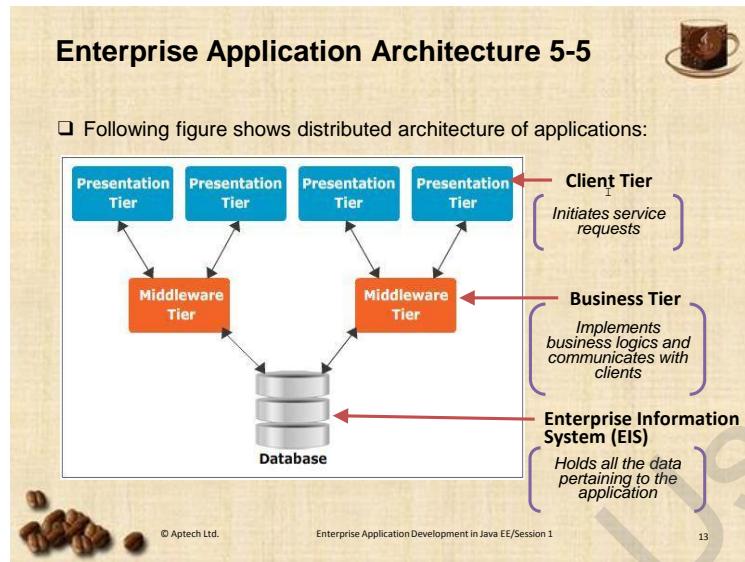
These functions are implemented as a separate layer leading to three-tier architecture.

Additional References:

You can prepare yourself with some additional content on distributed architectures by following this link: <http://www.developer.com/design/article.php/3808106/Introducing-Enterprise-Java-Application-Architecture-and-Design.htm>.

Slide 13

Let us understand about putting the enterprise application in three-tier architecture.



Use slide 13 to explain the tiers in the multitier applications.

The Java EE platform uses a distributed multitier application model for enterprise applications. Application logic is divided into components according to function, and the various application components that make up a Java EE application are installed on different tiers in the multitiered Java EE environment to which the application component belongs.

As shown in the figure given on slide 13, the multitiered Java EE application divides the application components into the different layers.

These layers of the distributed architecture are namely, presentation tier, middleware tier, and database tier.

- Presentation tier comprises the client-side functionality of the application. All the application requests are initiated in this tier.
- The business logic of the application is implemented in the middleware tier. It comprises all the classes and objects created in the application. Middleware can communicate with both the presentation tier and enterprise information system tier. It accesses the database as per requirement.
- Database tier comprises all the application data. It is the database of the application. Database can be implemented through technologies such as Oracle, Microsoft SQL Server, and so on. Middleware tier can use Java technology such as JDBC to connect to the database tier.

The distributed architecture enables development of components that can be distributed on the multiple layers and can be accessed on the network through appropriate protocols. This architecture improves the efficiency of the application and makes the remote location of the components transparent to the end user of the application.

The distribution of components on different layers has further evolved into n-tier architecture.

Tips:

Java EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end storage.

Additional References:

To know more about Java EE multitiered architecture, you can visit this link:
<http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html>.

Slide 14

Let us understand the different features of distributed application architecture.

Requirements of Distributed Applications

- Reusability
- Remote methods
- Resource pooling
- Multi-user
- Access control
- Failover support
- Transactional
- Shared data
- Logging and auditing
- Security

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 14

Use slide 14 to explain different features of distributed applications.

Discuss various issues or concerns that need to be considered and solved with respect to the design of the distributed applications.

Enterprise applications that are based on distributed architecture must fulfil some of the following requirements:

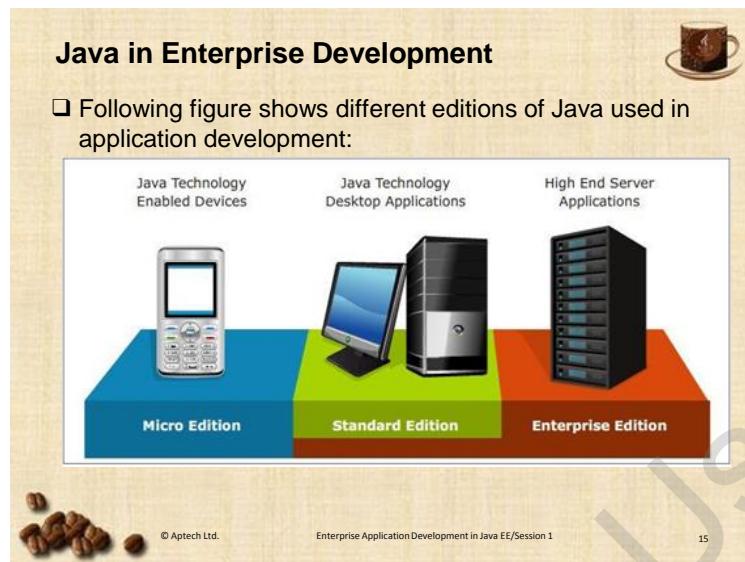
1. **Reusability** - Reusability specifies that the same business object may be used by different sub-systems of the same application as well as by different applications. For instance, both order management and invoicing applications may make use of the same product business object.
2. **Remote access** - Various components of the enterprise applications can interact with each other remotely. Since, distributed architecture can place different components of the application at different locations, the components should be remotely accessible. Developers should define remote methods to access different application components. This in turn requires implementing functions of parameter passing, dispatching SQL commands, and so on.
3. **Resource pooling** - Distributed architecture pools resources such as database connections, enterprise beans, and so on to improve the performance of the application. Resource pooling in distributed applications ensures that these resources are available.
4. **Multi-user** - A business object should be able to provide its services to a large number of clients at the same time.
5. **Access control** - The access to different resources of the application has to be well defined, the application should have well-defined roles and credentials should be appropriately assigned to each role. A manager for example can view as well as modify a business object and a junior-level operator may just view the business object.
6. **Failover support** – When a distributed application is deployed on a set of application servers, then the failure of one of the application servers should not be visible to the end

user of the application. All the requests being processed by failed server are transferred to another server in the cluster. Distributed architecture uses measures such as server clustering to implement failover support. When one of the servers in the cluster fails, then the requests coming from the clients are transferred to the remaining servers in the cluster.

7. **Transactional** -Business objects often need to participate in transactions. A transaction can be described as a set of functions that need to be completed as a unit. If one of the functions fails, all the functions in the unit will be cancelled or rolled-back. If all of them succeed, the transaction is said to be committed.
8. **Shared data** - The data of the application is shared by different components and may be simultaneously accessed by different application components, in such a scenario the integrity of the application data has to be protected. Distributed application should implement appropriate mechanisms for this purpose.
9. **Logging and auditing** - It is implemented in enterprise applications to keep track of each action that has been executed by the application. Application logging and auditing enables restoring the system in case of system failure and also tracks the application performance.
10. **Security** – Security implementation of distributed application is crucial, as it is accessed by different users. In distributed applications, since the components are remotely accessed, the components should be secured from malicious users.

Slide 15

Let us understand different versions of Java.



Use slide 15 to explain the different editions of Java and usage of each of these versions for different types of applications.

Java provides three editions of the platform – Java Standard Edition (Java SE), Java Micro Edition (Java ME), and Java Enterprise Edition (Java EE).

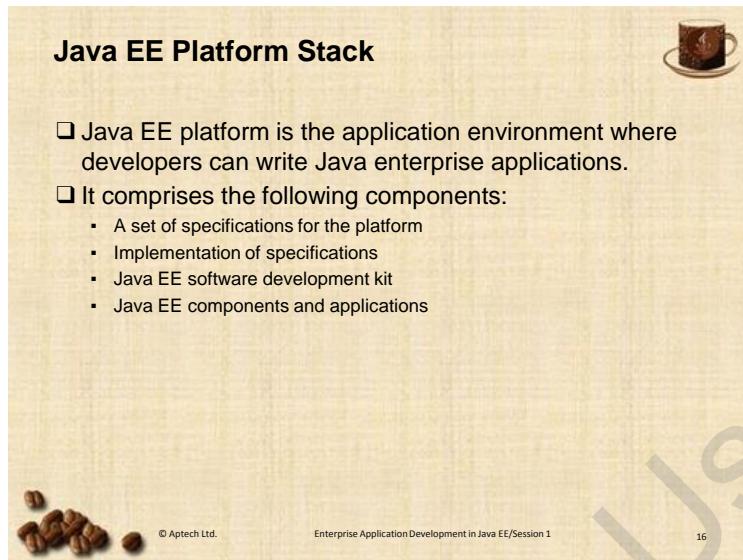
Java SE has the basic functionality and is used for developing desktop applications. Apart from the core API, Java SE has various development and deployment tools, virtual machines, and class libraries.

Java ME is a minimal edition of Java and is used to implement applications for devices with limited resources such as handheld devices, mobile phones, and so on.

Java EE is the enterprise edition. It is used to implement large enterprise applications. This is the most elaborate version which can be used to develop and implement huge applications. The purpose of Java EE is to build large scalable, multi-user applications. Though, the basic programming tools are the same, the tools in Java EE are meant for larger applications. Applications developed using Java EE are also known as enterprise applications.

Slide 16

Let us understand Java EE and its components.



The slide has a title "Java EE Platform Stack" at the top left. On the right side, there is a small graphic of a coffee cup with steam rising from it. At the bottom left, there is a small illustration of coffee beans. The slide contains the following text:

- ❑ Java EE platform is the application environment where developers can write Java enterprise applications.
- ❑ It comprises the following components:
 - A set of specifications for the platform
 - Implementation of specifications
 - Java EE software development kit
 - Java EE components and applications

At the bottom center, there is a copyright notice: © Aptech Ltd. Enterprise Application Development in Java EE/Session 1. At the bottom right, there is a page number: 16.

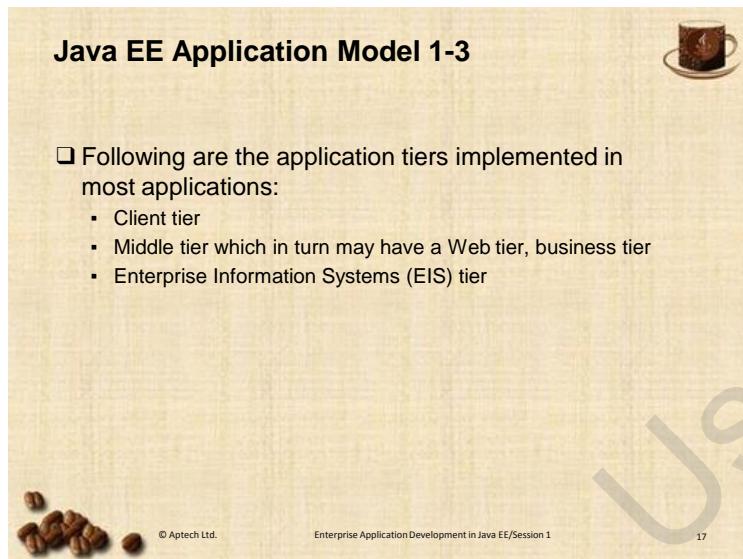
Use slide 16 to explain Java EE features and its components.

Some of the features of Java EE platform stack are as follows:

- **A set of specifications for the platform** – This is a standard defined by the architects of Java. The specifications of the Java EE platform are defined in terms of architecture, security, transaction management, resources, Application Programming Interfaces (APIs), interoperability, and so on.
- **Implementation of specifications** - The specification only defines the standard way in which the component can be implemented. Actual implementation of the specification has to be developed. These implementations can further be open source or proprietary products. The implementations are supported through a set of tools required for managing and monitoring the implementation.
- **Java EE Software Development Kit** – The Java EE SDK provides a basic set of necessary implementations and tools required to develop and build a Java EE application. The development kit includes tools such as Java compiler and interpreter.
- **Java EE components and applications** – The Java EE platform is built to develop and build enterprise application and its components. Java EE platform provides APIs and services to create and manage different application components.

Slide 17

Let us understand the multi-tier architecture supported by Java EE.



The slide has a parchment-like background with a coffee cup icon in the top right corner and coffee beans at the bottom left. The title "Java EE Application Model 1-3" is at the top left. The content area contains a question and a list:

❑ Following are the application tiers implemented in most applications:

- Client tier
- Middle tier which in turn may have a Web tier, business tier
- Enterprise Information Systems (EIS) tier

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 17

Use slide 17 to explain the n-tier architecture supported by Java EE application model.

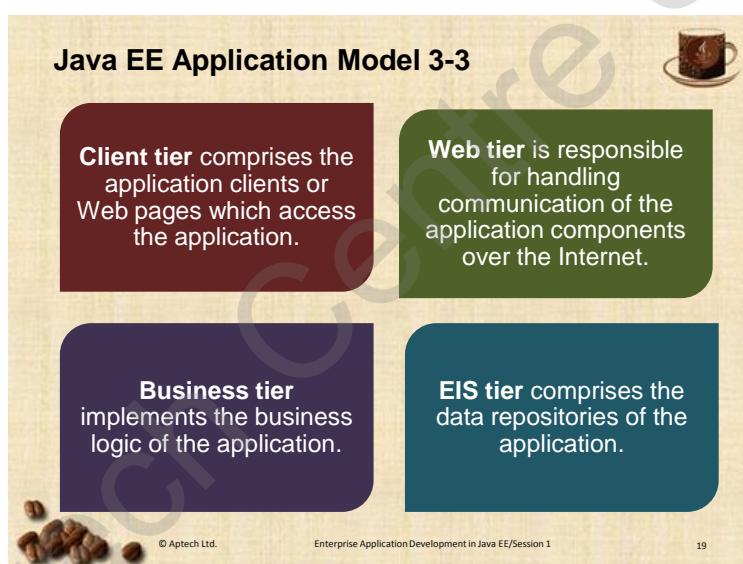
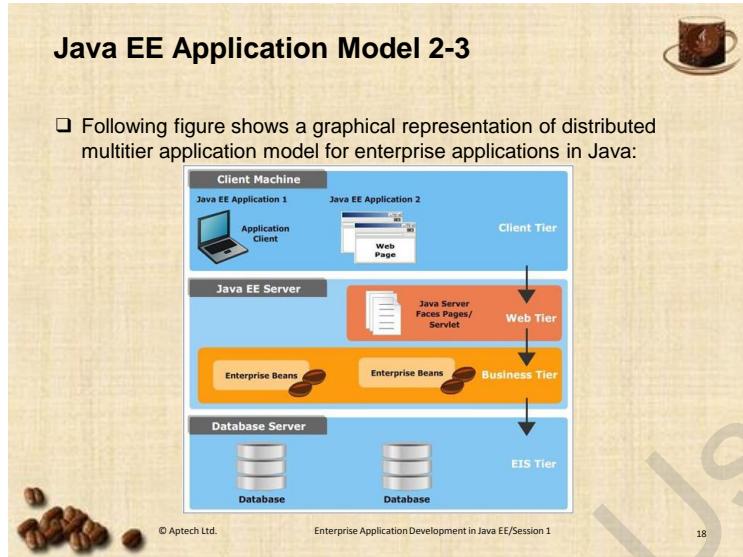
As discussed in the earlier slides, the n-tier architecture divides the functionality of the application into n layers. As in case of three-tier architecture, the implementation of enterprise application functions is divided into three layers.

Then, explain the type of tiers created for Java EE application model. There are normally these tier in an enterprise application:

- Client tier
- Middle tier which contain a Web tier and business tier
- Enterprise Information System tier

Slides 18 and 19

Let us understand the different tiers of Java applications.



Use slides 18 and 19 to discuss the functionality performed by the different tiers of an enterprise application.

Following are the functions of each of the tiers:

Use the figure given on slide 18 to explain the components developed for each tier. Use slide 19 to explain the functions implemented at each tier in an enterprise application.

Client tier comprises the application clients or Web pages which access the application. End users of the application access it through the client tier. A request is initiated from the client tier which is further forwarded to the business tier in case of enterprise applications and Web tier in case of Web applications. The client tier also receives the response from the business tier and returns it to the end user.

Web tier as the name suggests is responsible for handling communication of the application components over the Internet. The Web tier contains Java EE technologies such as JavaServer Faces (JSF), Servlets, and JavaServer Pages (JSP).

Business tier of the application implements the business logic of the application. This tier is customized and developed according to the application requirement. The technologies used in business tier are Enterprise Java Beans (EJB) with respect to enterprise applications, Java Restful (JAX-RS) Web services, and Java Persistence API (JPA) entities.

Enterprise Information Systems tier comprises the data repositories of the application that is, the database servers and legacy systems such as mainframes. The EIS tier is accessed through the business tier of the application. It cannot be directly accessed from the client tier. The Java EE technologies used in this tier are Java Database Connectivity (JDBC), Java Persistence API, Java EE Connector Architecture (JCA), and Java Transaction API (JTA).

Slides 20 to 23

Let us understand the functionality of EJBs as business components.

EJB as Business Components 1-4



- EJBs are server-side components.
- EJBs can communicate with both the end user and the database of the application.
- Processing of data is defined by the business requirements of the application and implemented through EJBs in case of Java.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 20

EJB as Business Components 2-4



- Following are the functions that can be carried out by EJBs:

 Implements business logic	 Access Database
 Integrates with other systems	 Enables deployment and execution of business components in distributed, multi-user environment

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 21

EJB as Business Components 3-4



- Remote components were earlier accessed through Remote Method Invocation (RMI).
- Following are the steps based on which RMI is implemented:
 - When a component has to access another component located over the network, the client invokes a stub.
 - Stub invokes the component through a skeleton.
 - Skeleton extracts the parameters from the request and invokes right methods to generate the response for the request received.
 - Once the response is generated, the response is sent to the client.
- Some drawbacks of RMI are loss of object identity which creates performance bottlenecks and so on.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 22

EJB as Business Components 4-4

The diagram illustrates the EJB architecture as business components. It shows a Server side and a Client side connected via a Network. The Server side contains a yellow box labeled 'Server' with three stacked components: 'Interface' (purple), 'Implémentation' (blue), and 'Skeleton' (orange). The Client side contains a yellow box labeled 'Client' with two stacked components: 'Interface' (purple) and 'Stub' (orange). Arrows indicate the flow from the Server's Interface to the Client's Interface, and from the Client's Stub back to the Server's Skeleton.

□ EJB overcomes the drawbacks of RMI and other technologies like Common Object Request Broken Architecture (CORBA) through component and container framework model.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 23

Use slides 20 to 23 to explain different functions of EJBs business components.

Use slide 20 to explain the requirement of EJB components. EJBs are server-side components in the application. An enterprise application has a user interface which can be application clients, Web pages, and so on through which the end user puts the application to utility. The data received from the user has to be processed by the application and then, either stored on the database or returned to the client. Processing of data is defined by the business requirements of the application and implemented through EJBs in case of Java. EJB's can communicate with both the end user and the database of the application.

Use slide 21 to explain the operations implemented by EJBs. EJBs are accessed by the application clients. They can access the application database, enable different components of the application to integrate with each other, and function together.

EJBs enable deployment and execution of application business components in a multi-user environment.

Use slide 22 to explain Remote Method invocation (RMI).

Tell them the working of RMI has already been covered in the earlier slides, so in this slide, you brief out the RMI mechanism.

Discuss with them about their understanding on RMI and its drawbacks. Tell them RMI is based on Java objects. The remote interface accessed by the RMI clients' needs to know the location of the server RMI object for communication.

The communication between the client and the stub requires a dedicated connection; any loss of network connection will reset the communication. Another disadvantage of RMI is that both the communicating entities should use Java platform.

Since the business methods are invoked over the network, security of the data is also a major issue with RMI. The distributed components combine the functionality or characteristics of components with the middleware systems to provide inter-process communication between the components.

The distributed component technologies such as EJB and CORBA are server-side technologies used for developing distributed business objects. The CORBA component model was developed to provide a distributed component model for use with programming languages other than Java and at the same time achieve interoperability with EJB components.

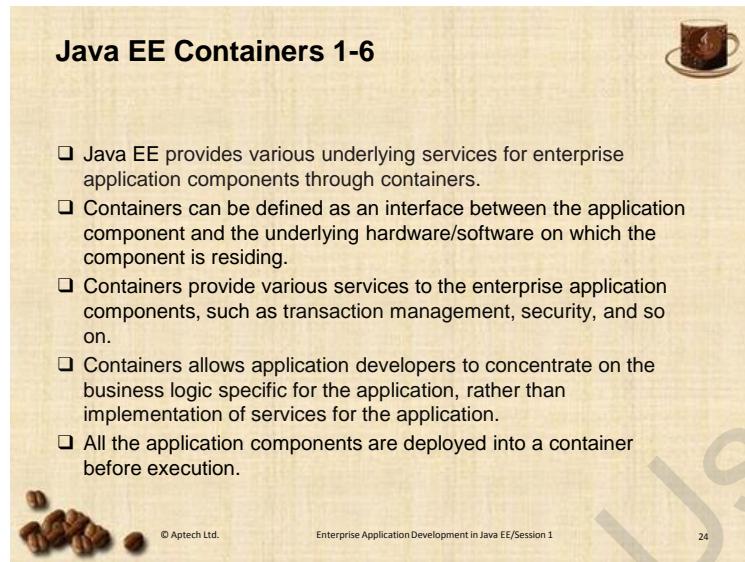
EJB tends to better support the deployment environment with the strong notion of configuration. While CORBA does not say much about configuration. EJB provides a clear separation of interface development such as remote interface to the developer.

Additional References:

To know how to access EJB components from non-Java based applications, you can check this link:
<http://www.javaworld.com/article/2074144/jndi/integrate-ejbs-with-corba.html>.

Slide 24

Let us understand containers in Java EE applications.



The slide has a light beige background with a faint watermark of coffee beans and a cup of coffee on the left side. At the top center, the title "Java EE Containers 1-6" is displayed in bold black font. In the top right corner, there is a small icon of a coffee cup. At the bottom left, there is a small illustration of coffee beans. The bottom right corner contains copyright information: "© Aptech Ltd.", "Enterprise Application Development in Java EE/Session 1", and the page number "24".

- ❑ Java EE provides various underlying services for enterprise application components through containers.
- ❑ Containers can be defined as an interface between the application component and the underlying hardware/software on which the component is residing.
- ❑ Containers provide various services to the enterprise application components, such as transaction management, security, and so on.
- ❑ Containers allow application developers to concentrate on the business logic specific for the application, rather than implementation of services for the application.
- ❑ All the application components are deployed into a container before execution.

Use slide 24 to explain what containers are and how they are used in enterprise applications.

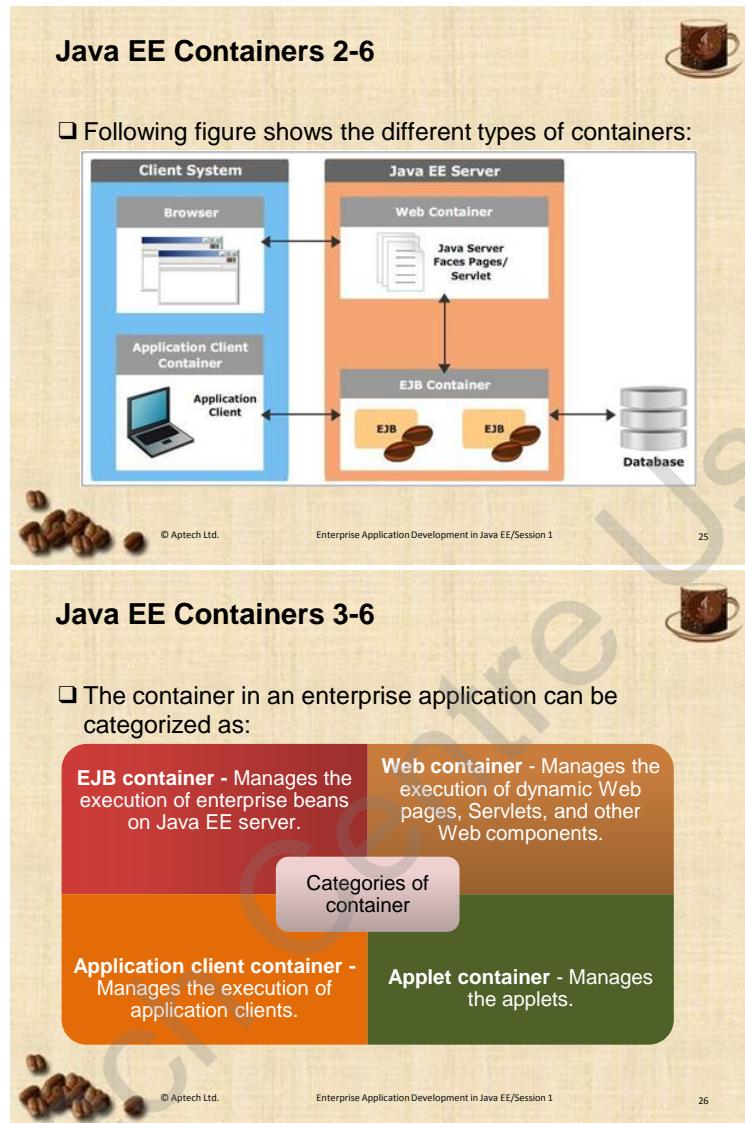
Irrespective of the application domain, there are certain generic tasks which are required for the implementation of most enterprise applications such as application security, transaction management, database connections, and so on. Containers implement these functions and provide them as a service to the applications deployed in it.

This enables the developers to focus on the business logic which is specific to the current application rather than spend time in implementing the mentioned generic tasks. Application development process thus, becomes more efficient and quick.

All the application components on the application server are deployed in a container. Any request to the application on the server first reaches the container. The container then decides which of the enterprise bean has to be invoked to service the request. Whenever, the EJB requires accessing a database it provides the request to the container, which in turn performs the database access operations.

Slides 25 and 26

Let us understand how the application components are deployed into different containers.



Use slides 25 and 26 to explain how containers function on the application server.

The figure shows on slide 25 depicts the containers provided in an application server. Normally, there are three containers in an application server. The first container is called as client container. The client container can contain components such as a Web client or an application client. Similarly, the Web container deploys Web components and EJB container deploys EJB business objects in an application server. The deployment process performed on the Java application server, installs the application components into appropriate container.

Tell them that depending on the server used which can be either application server or Web server, the number of container varies.

Use slide 26 to explain different types of containers in the Java EE application servers.

EJB container has all the EJBs of the application deployed in it. This container is on application server and provides security mechanisms, transaction management, database connectivity, and so on to the EJBs deployed in it.

Web container is also on the application server and it has the server-side components of the Web application deployed in it. The components deployed in a Web container are Servlets, JSFs, and so on. The Web container provides connectivity over the Internet through HTTP.

Application client container is present on the client-side. They are Java applications which has a set of supported libraries and executes in a Java SE environment.

Applet container manages the execution of applets. It consists of a web browser and Java Plug-in running on the client together.

In-Class Question:

After you finish explaining features and functionalities of containers, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



In which container are the servlets of an enterprise application deployed?

Answer:

Web container

Slides 27 to 29

Let us understand the services provided by the container.

Java EE Containers 4-6



- ❑ The services provided by the container to the application component are:
 - Security
 - Provides security model by configuring the container.
 - Transaction support
 - Provides transaction support to the application component.
 - Java Naming and Directory service
 - Provides naming services to ensure that the objects of the application are appropriately accessed.
 - Java EE communication model
 - Provides low level communication between clients and enterprise beans.

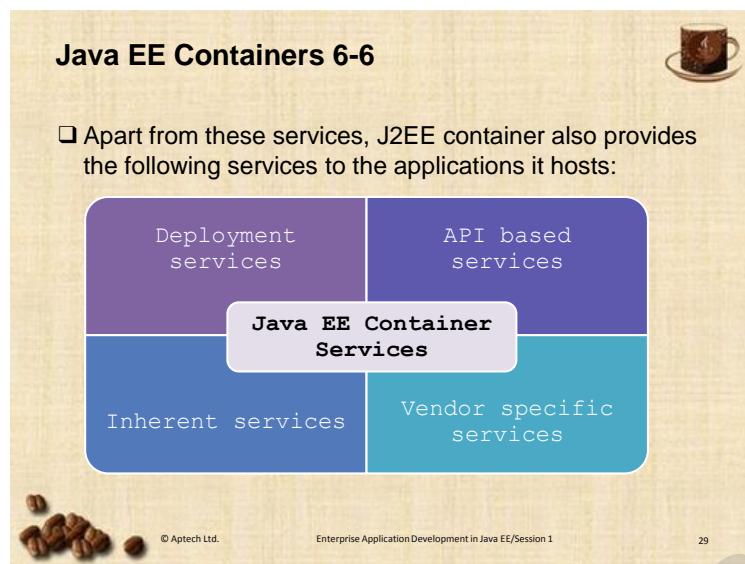
© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 27

Java EE Containers 5-6



- ❑ The containers also manages enterprise bean and Servlet lifecycle, database connection, resource pooling, data persistence, and access to the Java EE platform APIs.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 28



Use slides 27 to 29 to explain different services the container provides for the components deployed in it.

Security – Java EE provides for security model by configuring the container. The authentication and authorization mechanisms can be configured in the container while deploying the application. Being a distributed application, implementing security mechanisms is essential for all the enterprise applications. These mechanisms can be uniformly defined for all the components of the enterprise application by the container.

Transaction support – The container provides transaction support to the application component. The Java EE transaction model links various methods which execute as a part of transaction and treats them as a single unit. Transaction management is crucial for integrity of the data sources of the application. The locking mechanisms and the isolation levels of the transaction can be defined for the entire application through the deployment descriptor. The container accesses the deployment descriptor to define the transaction management model of the application.

Java Naming and Directory Interface (JNDI) - Provides naming services through the container to ensure that the objects of the application are appropriately accessed. JNDI is a service of Java EE to simplify access to different application components by providing a uniform naming pattern.

The Java EE communication model - It is implemented by the container to provide low-level communication between clients and enterprise JavaBeans.

The container also manages enterprise bean and servlet lifecycles, database connection, resource pooling, data persistence, and access to the Java EE platform APIs. The lifecycle of these entities begin with instantiating the component, loading it into the container and using it in processing client requests. After utility of the component is complete, all the resources allocated for the component has to be deallocated, which ends the lifecycle of the component.

Use slide 29 to explain some more services provided by the container.

Deployment services refer to a category of services which enable application deployment. The application deployment is done based on the deployment descriptor of the application. The deployment descriptor files are also referred as configuration files. Every enterprise application has at least one deployment descriptor. For Web-based applications, it is `web.xml` and for business objects, it is `ejb-jar.xml`. Developers have to define all the application configuration details in the deployment descriptor. The container uses this deployment descriptor and appropriately configures the application.

API based services refer to the services provided by the container to support the Java APIs used in the application. Java EE provides APIs to implement functions such as mailing, messaging, persistence, transaction management, and so on in the application. Containers provide services for these functions according to the application configuration.

Inherent services are those services which manage the lifecycle of the components deployed in the container.

Vendor-specific services provided by the container are those which are essential for the application end user. These are services such as load balancing, scalability, high availability, and so on.

In-Class Question:

After you finish explaining the services provided by the containers, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which container service provides access to the business components on the client-side?

Answer:

Java Naming and Directory Interface (JNDI)

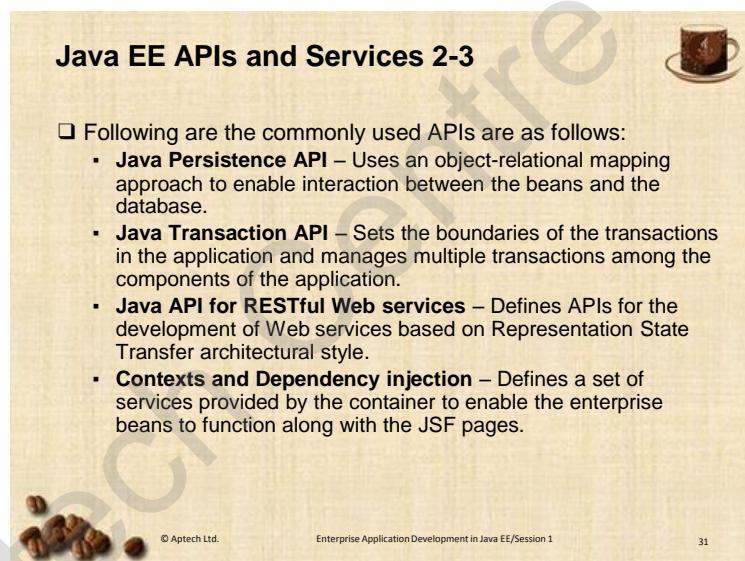
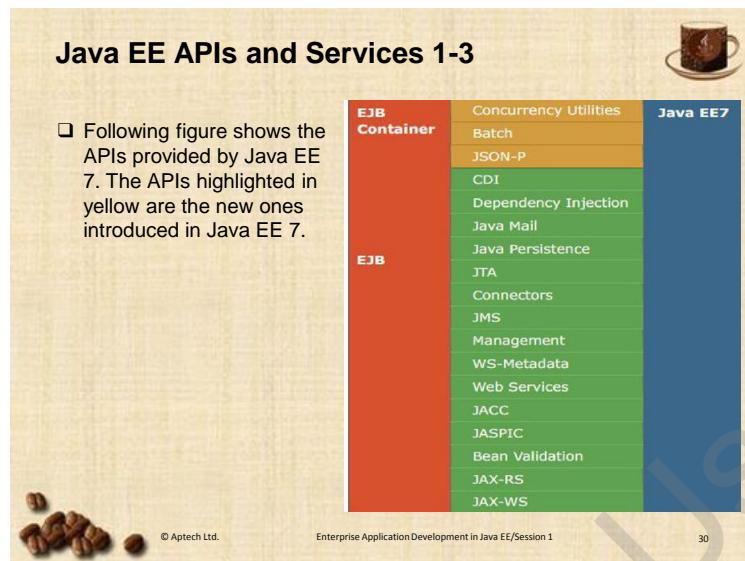
Additional References:

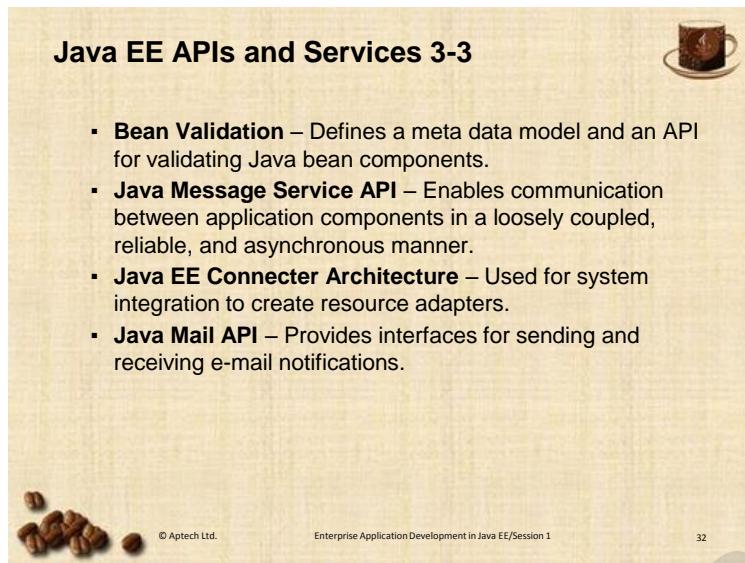
For more information on Java EE application containers, you can refer this link:

<http://docs.oracle.com/javaee/5/tutorial/doc/bnabo.html>.

Slides 30 to 32

Let us understand the APIs provided by the Java EE platform.





Java EE APIs and Services 3-3

- **Bean Validation** – Defines a meta data model and an API for validating Java bean components.
- **Java Message Service API** – Enables communication between application components in a loosely coupled, reliable, and asynchronous manner.
- **Java EE Connector Architecture** – Used for system integration to create resource adapters.
- **Java Mail API** – Provides interfaces for sending and receiving e-mail notifications.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 32

Use slides 30 to 32 to explain different APIs provided by Java EE 7.

Tell the students that most of these APIs were present in earlier versions of Java EE 7. The APIs highlighted in yellow have undergone major modifications in the current version of Java EE. Concurrency utilities in Java EE 7 provide asynchronous capabilities for application components.

Batch jobs are application tasks which are to be executed without user interaction. The Batch Applications for Java EE 7 provide support for creating and running batch jobs.

JSON is a text-based data exchange format. It is based on JavaScript which is used in Web services. Java EE 7 enables JSON processing in applications.

Use slides 31 and 32 to introduce different APIs in Java EE. Tell the students that this is just an introduction and that these APIs will be discussed in depth in later sessions.

- **Java Persistence API** – The persistence API uses an object-relational mapping approach to enable interaction between the beans and the database. This API is used when the applications require some data to be stored on permanent storage.
- **Java Transaction API** – The transaction API sets the boundaries of the transactions in the application and manages multiple transactions among the components of the application. In multi-user systems when different operations are performed by multiple users the access should be moderated. Transaction management moderates these operations performed to ensure data integrity.
- **Java API for RESTful Web Services** – Defines APIs for the development of Web services based on Representation State Transfer architectural style.
- **Contexts and Dependency Injection** – This defines a set of services provided by the container to enable the enterprise beans to function along with the JSF pages.
- **Bean Validation** – This defines a Meta data model and an API for validating Java bean components.
- **Java Message Service API** – Enables communication between application components in a loosely coupled, reliable and asynchronous manner. When an application requires to send

messages to its clients, developers can use this API to implement this function in the application.

- **Java EE Connector Architecture** – It is primarily used for system integration to create resource adapters that access the enterprise information system. When there are different application components, for instance an Oracle database which interprets data in the form of tables, rows and columns and EJB should interact with this component, the EJB interprets data in the form of objects. The gap of interpreting data by the two components should be bridged. Java EE provides Connectors to serve this purpose.
- **Java Mail API** – The Java Mail API provides interfaces for sending and receiving email notifications. The Java Mail API has two interfaces – application level interface and a service provider interface.

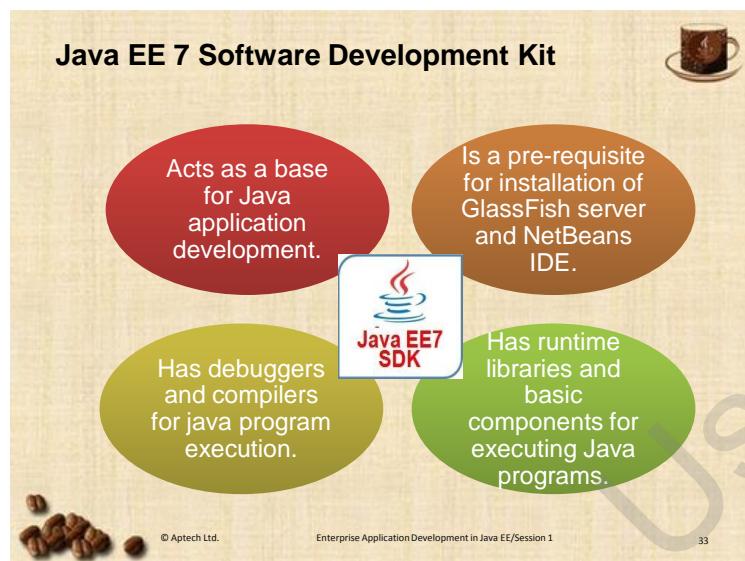
Additional References:

To know about Java EE APIs for Web and EJB container, you can refer this link:

<https://docs.oracle.com/javaee/7/tutorial/overview007.htm>.

Slide 33

Let us understand the tools present in the Java EE 7 SDK.



Use slide 33 to explain the development tools provided by Java EE 7 SDK.

Java EE SDK comprises basic set of tools required for application execution. However to execute a basic enterprise application, an application server is essential. For example, the Java EE 7 SDK acts as a base for Java application development. This is a pre-requisite for installation of GlassFish server and NetBeans IDE.

Enterprise applications are usually developed in IDE. IDE provides a skeletal structure of the application according to which the developer can create the application components.

The SDK has the compiler, interpreter, and other runtime libraries required for executing Java programs.

Slides 34 to 36

Let us understand application servers which are used for Java enterprise applications.

Java EE Application Servers 1-3

Application servers:

- Are entities of applications on which enterprise applications are deployed and run.
- Consists of components such as database connectors, Web server connectors, runtime libraries, and so on.
- Comprise an EJB container and Web container.
 - All beans of an application are deployed in the EJB container.
 - Web components are deployed in the Web container.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 34

Java EE Application Servers 2-3

Following are the Java EE application servers:

- WildFly
- IBM WebSphere
- GlassFish
- Apache TomEE
- Oracle WebLogic Server

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 35

Java EE Application Servers 3-3

Java EE application is done in NetBeans IDE.

NetBeans IDE is an integrated development environment used for development in Java.

Following figure shows the NetBeans IE interface:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 36

Use slides 34 to 36 to explain the significance of application servers in enterprise applications.

Use slide 34 to explain application server. Application servers are entities of applications on which enterprise applications are deployed and run. Application servers consist of components such as database connectors, Web server connectors, runtime libraries, and so on.

All the components of the enterprise application are deployed on different containers on the application server. The application server comprises an EJB container and Web container. All the bean components of the application are deployed in the EJB container and all the Web components of the application are deployed in the Web container.

Use slide 35 to introduce different application servers which can be used to deploy enterprise applications. Tell the students that some of these servers are open source such as GlassFish, and Apache, while others are proprietary such as Oracle Weblogic server.

GlassFish is the reference implementation of Java EE and supports Enterprise Java Beans, JavaServer Faces (JSF), Java Persistence API (JPA) and Java Messaging Service (JMS). In the current course, GlassFish 4.0 is being used along with NetBeans IDE.

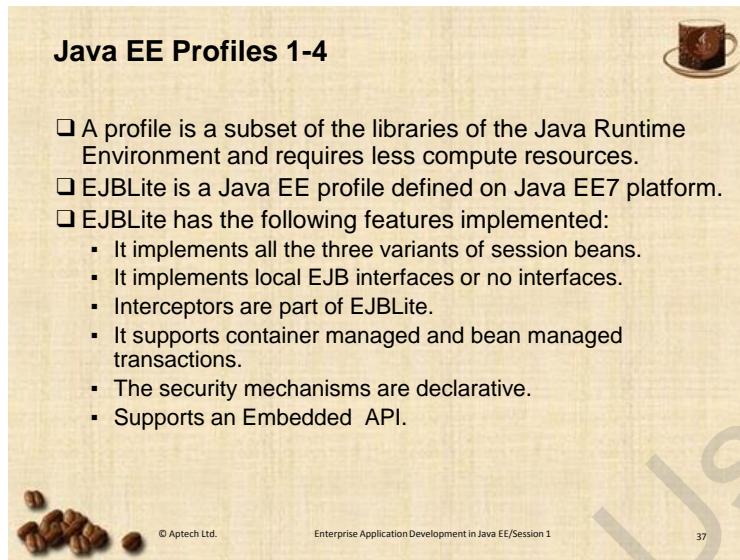
Use slide 36 to explain what NetBeans IDE is and how it is used for Java enterprise application development.

The IDE provides comprehensive tools for software development. The developers can edit code, build code and debug through simple interface provided by the IDE. There are several IDEs available for different technologies. Eclipse and NetBeans are IDEs which are compatible for Java application development.

NetBeans with its code analyzers, editors, and converters provides best support to Java EE application development. Apart from coding it also provides interface for project management which can be used by Project Managers and Team Leaders in an organization. NetBeans IDE also enables code refactoring where it prompts the developer with the probable methods that they can use from certain class.

Slide 37

Let us understand Java EE profiles.



Java EE Profiles 1-4

- ❑ A profile is a subset of the libraries of the Java Runtime Environment and requires less compute resources.
- ❑ EJBLite is a Java EE profile defined on Java EE7 platform.
- ❑ EJBLite has the following features implemented:
 - It implements all the three variants of session beans.
 - It implements local EJB interfaces or no interfaces.
 - Interceptors are part of EJBLite.
 - It supports container managed and bean managed transactions.
 - The security mechanisms are declarative.
 - Supports an Embedded API.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 37

Use slide 37 to explain Java profiles provided by Java EE.

Every application does not require the entire Java EE for application development, therefore Java EE defines Java profiles which have a sub-set of features provided by Java EE 7.

EJBLite is one such profile. As the name suggests EJBLite is a lighter version of Java EE. EJB Lite meets the needs of these applications with a small subset of the features in EJB 3.1 centered around the session bean component model.

Explain the features present in EJBLite and highlight how it is different from Java EE. It is essential to highlight how certain feature makes it lighter than complete Java EE. Use the tables provided in the slide to explain the features provided in EJBLite.

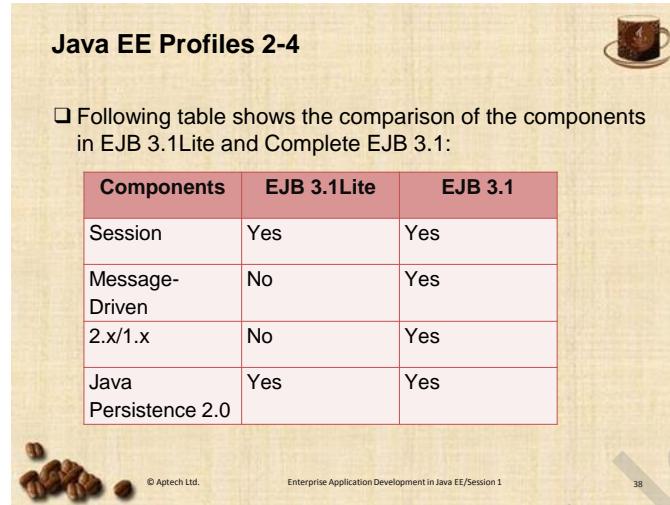
The security mechanisms in EJBLite are declarative, which implies that the developer cannot define his/her own security mechanism which makes the application simpler.

Embedded APIs are used to reduce the memory consumption of the application. Java provides various embedded APIs along with Java ME. In case of Java EE profiles, few of these APIs are included. In EJB 3.1, the EJBLite contains an Embedded API and container for use in Java SE environment. This makes the testing of EJB components easy to test outside the Java EE container.

Slides 38 to 40

Let us compare the different features of EJBLite with EJB.

Java EE Profiles 2-4

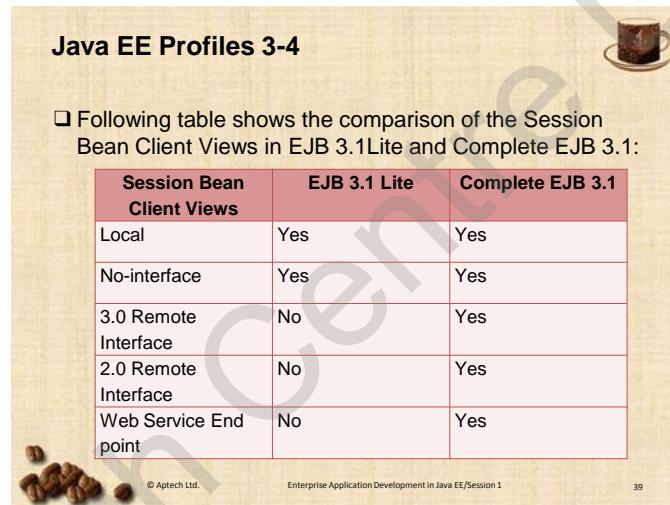


□ Following table shows the comparison of the components in EJB 3.1Lite and Complete EJB 3.1:

Components	EJB 3.1Lite	EJB 3.1
Session	Yes	Yes
Message-Driven	No	Yes
2.x/1.x	No	Yes
Java Persistence 2.0	Yes	Yes

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 38

Java EE Profiles 3-4

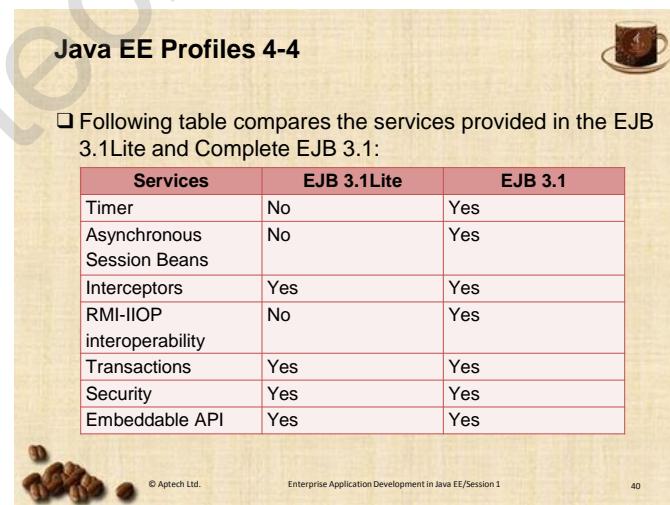


□ Following table shows the comparison of the Session Bean Client Views in EJB 3.1Lite and Complete EJB 3.1:

Session Bean Client Views	EJB 3.1 Lite	Complete EJB 3.1
Local	Yes	Yes
No-interface	Yes	Yes
3.0 Remote Interface	No	Yes
2.0 Remote Interface	No	Yes
Web Service End point	No	Yes

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 39

Java EE Profiles 4-4



□ Following table compares the services provided in the EJB 3.1Lite and Complete EJB 3.1:

Services	EJB 3.1Lite	EJB 3.1
Timer	No	Yes
Asynchronous Session Beans	No	Yes
Interceptors	Yes	Yes
RMI-IIOP interoperability	No	Yes
Transactions	Yes	Yes
Security	Yes	Yes
Embeddable API	Yes	Yes

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 40

Use slides 38 to 40 to highlight the differences between EJBLite3.1 and EJB3.1.

Slide 41

Let us summarize the session.

Summary

- ❑ An enterprise application is a large business application. It is usually hosted on servers and simultaneously provides services to a large number of users over a computer network.
- ❑ Application architecture divides application components into tiers or layers based on their functionality in the environment.
- ❑ Distributed application architecture divides the large monolithic application into layers.
- ❑ As enterprise application execution is distributed across multiple tiers, there are various issues or concerns with respect to the design of these applications.
- ❑ The purpose of Java EE platform is to build large scalable, multi user applications.
- ❑ The business logic of an application is implemented by EJBs by using various supporting APIs provided by Java EE.
- ❑ EJB is based on component framework model where the application comprises communicating components.
- ❑ Enterprise application components are deployed into containers that provide supporting services such as transaction management, security, and so on to the application.
- ❑ Application servers are entities of applications on which enterprise applications are deployed and run.
- ❑ EJBLite is a Java EE profile, which is a compact runtime environment including all the essential application entities.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 1 41

Using slide 41, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

1.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the component-based model and use of enterprise beans in developing business objects for Java enterprise applications.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 2 – Enterprise JavaBeans

2.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

2.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the principles of component-based development
- Define Enterprise JavaBeans
- List the characteristics of enterprise JavaBeans
- Describe the evolution of enterprise JavaBeans
- Explain the features of EJB 3.0
- Explain the different types of Enterprise JavaBeans
- Explain the JNDI service on Java EE platform
- Explain JNDI APIs
- Describe the various roles involved in EJB application development
- Explain the various steps involved in developing and packaging an enterprise application
- Describe tools used for developing enterprise application

2.1.2 Teaching Skills

To teach this session successfully, you should be aware of the usage of Enterprise JavaBeans (EJBs) in the enterprise application development. You should be aware of the characteristics of EJB, different types of EJBs, and their utility in application development.

You should aware yourself with Java Naming and Directory Interface (JNDI) service to access the beans remotely by the clients. You should also be aware of various steps in enterprise application development along with their packaging, assembling, and deploying on Java application servers.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❑ Describe the principles of component-based development
- ❑ Define Enterprise JavaBeans
- ❑ List the characteristics of enterprise JavaBeans
- ❑ Describe the evolution of enterprise JavaBeans
- ❑ Explain the features of EJB 3.0
- ❑ Explain the different types of Enterprise JavaBeans
- ❑ Explain the JNDI service on Java EE platform
- ❑ Explain JNDI APIs
- ❑ Describe the various roles involved in EJB application development
- ❑ Explain the various steps involved in developing and packaging an enterprise application
- ❑ Describe tools used for developing enterprise application

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 2

Tell them that they will be introduced to Enterprise JavaBeans (EJBs) and their characteristics in developing Java enterprise applications. The session explains about three variants of the EJBs which are used for processing business logic and enterprise-level data in enterprise applications.

The session explains how to register the components using Java Naming and Directory Interface (JNDI) on the application server to be accessed remotely by the clients. The session describes various tools used for developing EJB components and packaging of EJBs components on the Java EE platform.

2.2 In-Class Explanations

Slide 3

Let us understand the business components used in enterprise applications.

Introduction 1-2

- ❑ Enterprise applications are developed to fulfil the needs of domains such as banking, finance, and so on involving:
 - Large numbers of users
 - Complex requirements

- ❑ Enterprise application uses Enterprise JavaBeans (EJBs) as business components on the middleware tier of the multilayered application.

© Aptech Ltd. Enterprise Application Development in Java EE / Session 2 3

Use slide 3 to explain the role of business components in the enterprise applications.

Tell them that enterprise applications are developed to fulfil the needs of a specific domains such as banking, finance, and so on. They can be used by large number of processes used in the enterprise application developed for the specific domain.

The enterprise application uses Enterprise JavaBeans (EJBs) as business components on the middleware tier of the multilayered application architecture. The business components are used to implement the business logic of the application. The implementation of EJBs on the middleware tier is laid by the **EJB specification** provided by Sun Microsystems.

The EJB specification is one of the several Java APIs developed for the Java EE platform. Tell them that enterprise application development can be done on various platforms such as .NET, Java, and so on. Our focus of the course is Java EE 7.

In-Class Question:

After you finish explaining the business components in the enterprise applications, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are enterprise applications?

Answer:

Enterprise applications are real time applications which are implemented in a domain such as banking, hospital administration, customer management, and so on. Enterprise applications can manage huge volumes of data and multiple clients simultaneously.

Slides 4 to 6

Let us understand the implementation of EJBs through components.

Introduction 2-2



❑ The implementation of EJBs on the middleware tier is:

- Laid by the EJB specification provided by Sun Microsystems
- Based on **component-based** development framework.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 4

Principles of Component-Based Development 1-2



Component-Based Development

- They have an interface defined so that other application components can access it.
- They have a well defined lifecycle mechanism.
- They are configurable.
- They have a third-party integration scheme.
- Components can be assembled with other program blocks to build a complete application.
- Components are portable and reusable.
- Components can function independently or when assembled with other components.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 5

Principles of Component-Based Development 2-2



❑ Advantages of component-based applications are:

- **Reduction of cost and time in building large complicated systems:**
 - As there is a reuse of components.
- **Better quality of software is ensured:**
 - As the components are tested and stabilized through multiple iterations of testing.
- **A developer creating an application component does not require knowledge of the entire application:**
 - Since, the components are independently developed.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 6

Use slides 4 to 6 to explain component-based application development used in Java enterprise applications. Java EE enables applications to be developed as independent components with well-defined interfaces. These components are assembled together to function as a single application.

Each component of the application is defined in a way that it can independently be developed and tested, thus drastically reducing the application development time.

According to the EJB specification, applications are implemented as independent components which interact with each other. These components interact with each other in a loosely coupled manner to implement application related tasks.

Use slide 5 to explain the features of component-based application development.

- Every component of the application should have an interface defined. The interface defines the method of accessing the component by the outside world.
- Each component has a well-defined lifecycle. All the components of the application may not be always active. The components are activated and deactivated as per requirement. There must be a well-defined process in the application to activate and deactivate these components.
- Application components should be configurable. This is essential for assembling and integrating the application components. It is also required for tuning the application performance.
- The integration of the application is usually done by a different team which may not be involved in the application development process. The integration process is based on the interfaces and configuration of the components. The integration team need not know details about the internal functioning of the application.
- An application component used in certain application can be reused in other applications also, provided the components have compatible interfaces.

Use slide 6 to explain the advantages of component-based application development.

Component-based application development reduces the time of application development due to various factors:

- It allows parallel development of different components of an application, where independent teams can simultaneously develop each component.
- It provides for reuse of components, developers can pick up components from an application which was developed earlier and reuse it in the current application. The developer has to create remaining components and make them compatible with the existing components.

The application components which are taken from other applications are elaborately tested and hence, are stable components. This adds stability to the current application.

Since each component of the application is independently developed, a developer working on one component of the application need not have knowledge of other components of the application. A developer may be completely unaware about the functionality of other components of the application.

In-Class Question:

After you finish explaining implementation of EJB as component, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



How component-based development reduces the time of application?

Answer:

By parallel development of components and reuse of the components.

Slides 7 and 8

Let us look at the features of Enterprise JavaBeans.

Enterprise JavaBeans (EJBs) 1-2



- ❑ EJBs are server-side components of an enterprise application.
- ❑ EJB technology provides a platform for developing portable, reusable, and scalable business applications.
- ❑ EJB components are deployed into a container which provides generic services to these components.
- ❑ Containers services include such as security, persistence transaction management, and so on.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 2 7

Enterprise JavaBeans (EJBs) 2-2



- ❑ There are two ways of looking at EJBs:
 - **EJB are specifications**
 - Lay out rules and standards on how you should code your EJBs.
 - **EJBs are Java interfaces**
 - Are exposed to the EJB clients to access the implemented bean code.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 2 8

Use slides 7 and 8 to explain the features of Enterprise JavaBeans.

EJBs are server-side components which implement the business logic of the application. They are deployed in the container on application server. EJBs are used to process the client requests received by the application server.

EJBs enable creating portable, reusable, and scalable applications. Portability is an inherent feature of any Java application due to byte code and Java Virtual Machine. The components developed as a part of an application can be reused for other applications thus, enabling reusability. EJBs are instantiated into the container on the application server, in order to make the application scalable. The number of instances in the container can be increased and decreased as per application requirement.

Java EE provides basic functionalities required by the enterprise application through application container. This in turn enables the developers to focus on the business logic specific to the application, where the container provides the generic services to the application components.

The generic services provided by the container include security, transaction management, data persistence, and so on. These services are required by most of the enterprise applications. The Java EE platform provides these services through the container.

There are two ways of looking at EJBs:

- **EJBs as a set of specifications** – Developers can view EJBs as a set of specifications as it is an open source platform. Enthusiastic developers can use the specification provided by Sun Microsystems as a base and implement the specification. Developers can also customize the specification to suit their requirement.
- **EJBs as interfaces** – Clients of an enterprise application can access the service of the application through EJBs. Each EJB exposes a set of methods which can be invoked by the client. The methods exposed to the clients are known as the interface provided to the client.

Slide 9

Let us understand the characteristics of EJBs.

Characteristics of Enterprise Java Beans

- ❑ Following are the characteristics of Enterprise Java Beans:
 - Implements business logic of the application
 - Deployed in the EJB container on the Java-enabled application server
 - Configured through deployment descriptors which defines the configurations of the bean
 - Communicates with the application clients through interfaces

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 9

Use slide 9 to explain the characteristics of EJBS.

EJBs reside in the middle tier of the application. They can be accessed by the user interface in the presentation tier and in turn they can also access the database in the Enterprise Information System tier.

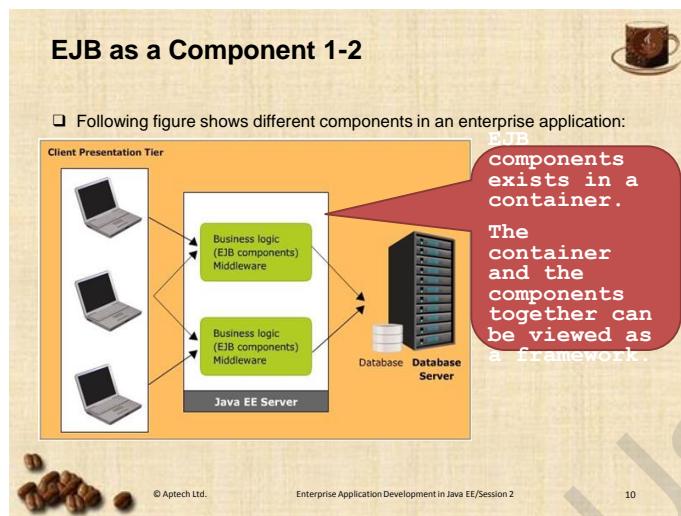
EJBs are deployed in the application container, also called as EJB container. The EJB container is a logical structure of Java EE platform which provides generic services to the EJB components deployed in it. The application container or EJB container is present on the application server.

Every enterprise application needs to be registered on the application server. This is done by specifying the information about EJB component in a deployment descriptor file. The deployment descriptor file is also called as the configuration file which keeps information of the application. In case of enterprise applications, it is `ejb-jar.xml` and in case of Web applications it is `web.xml`.

EJBs communicate with the application clients through the interface. They are also known as business interface. Business Interface refers to a set of methods which can be invoked by the application clients.

Slide 10

Let us understand how EJB as a component are placed in an enterprise application.



Use slide 10 to explain how EJBs as a component are placed in the enterprise application.

An application can be developed consisting of several reusable components. The main requirement of a component is that it should encapsulate the behavior of an application. Users are not aware of the internal processes of the components in an application, however, they are aware of what they need to pass in as input and what to expect as output.

Then, explain the figure as shown on slide 10. Tell them that the presentation layer contains the logic for displaying a user interface to the client, the middle-tier contains the actual business logic, and the database tier provides data services. The foundation of an application is the data that an application contains utilizes.

The data layer consists of a database and the access mechanisms used by the application for accessing the database. JDBC API is the low-level interface for accessing data from the relational databases. The application data on its own is not of any use till some processing logic is applied on it. The logic is applied in the business layer and the code in this layer maps to a company's business processes. For instance, a retail chain gives a 5% discount to each 100th customer in its stores. This type of requirement needs to be implemented in the business components running on the business layer.

The EJBs therefore sit behind presentation layer components and provide services to them. These presentation layer components are referred as clients. Java Applets and applications, Web pages, and Web services can act as EJB clients.

In-Class Question:

After you finish explaining the placing of EJB component you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which API is used for accessing data from the relational databases?

Answer:

JDBC API

Slide 11

Let us understand functions and usage of EJB component in an enterprise application.

EJB as a Component 2-2

- ❑ When an EJB is deployed in the EJB container, the EJB container creates multiple instances of the EJB to serve multiple clients.
- ❑ Collection of bean instances or beans in the EJB container is referred as bean pool.
- ❑ Following figure shows a container with bean pools:

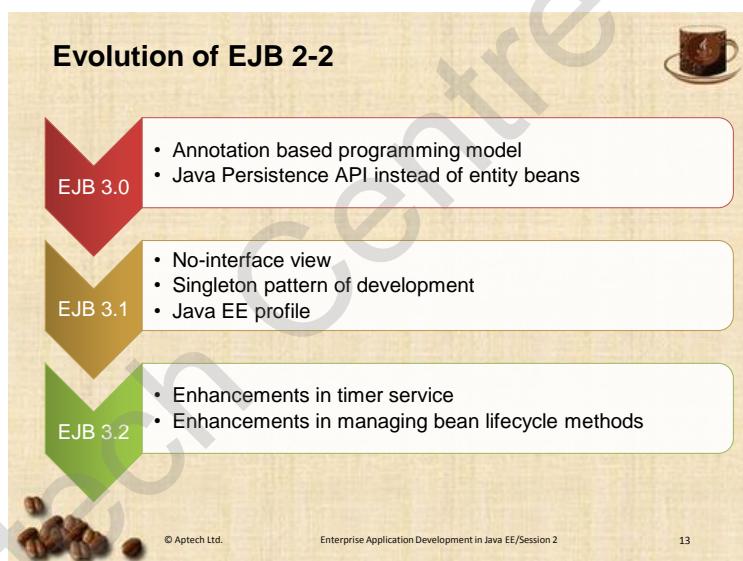
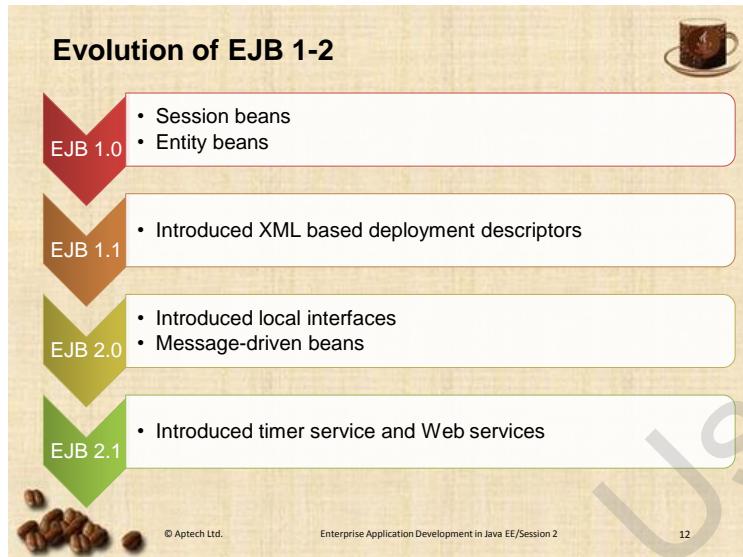
Use slide 11 to explain the functions and usage of the EJB components.

The EJB component is deployed in the container and it can be used to service different client requests. The lifecycle of the EJB component is managed by the container. When there are several requests for certain EJB component, then the container can create multiple instances of the EJB creating a pool of EJBs. Such a pool of EJBs is termed as a bean pool. Maintaining a pool of EJBs, improves the application performance.

The number of beans in the bean pool can be increased or decreased by the container based on the application requirement. The properties of the bean pool depend on the settings that are provided with the EJB during deployment. These settings pertain to the number of beans in the pool at a time, how many new beans should be added when the container runs short of beans, the maximum bean pool size, and so on. The container creates separate bean pools for each EJB.

Slides 12 and 13

Let us understand the evolution of EJB.



Use slides 12 and 13 to explain how EJB has evolved from EJB 1.0 to EJB 3.2.

Any software product evolves from version to version after initial release based on changing business requirements, changing hardware capabilities, and other factors. EJB has also evolved from EJB 1.0 to improve its performance and to adapt to the changing market needs.

EJB 1.0 had Session beans and Entity beans. Session beans are used to manage a series of requests and responses between the client and server for certain time duration. Entity beans of EJB 1.0 are used for implementing database related operations. Entity beans can connect with database, perform operations on the database, and modify the database.

EJB 1.1 introduced deployment descriptors. Deployment descriptors are declarative XML files which provide configuration for the enterprise application. An EJB application has an `ejb-jar.xml` which can be used by the deployer or by the deployment tool to appropriately deploy the application.

EJB 2.0 introduced local interfaces which refer a set of methods which can be accessed by local clients. This was not present in the earlier versions of EJB. EJB 2.0 also introduced message-driven beans which are used to process user messages. Message-driven beans cannot be explicitly invoked by the client. They are only invoked when the application receives a message. For instance, we have applications where you drop a mail to a bank or telecom service provider and you receive an automated response. This is an instance of message processing without human intervention.

EJB 2.1 introduced timer service which enables developers to automatically schedule application related tasks. A developer may intend to regularly perform application audits, measure the system performance, and so on. Such tasks can be scheduled through the timer service in applications.

EJB 3.0 introduced annotation-based programming. The deployment information of the enterprise application can be added by the developer in between the application code. Annotations also enable resource injection and dependency injection in the application. The Entity beans used in earlier versions for database operations were now replaced by Java Persistence API in EJB 3.0. JPA provides a simplified interface for the developers to access database.

EJB 3.1 improvised EJB 3.0 by providing no-interface view. No-interface view has all the public methods defined in the EJB. EJB 3.1 also introduced Java EE profiles which indicate a subset of the functionalities provided by Java EE. Profiles are defined to provide development platforms for lighter applications.

EJB 3.2 has enhanced the timer service, how the bean lifecycle methods are managed, also enhanced the activation, and passivation of Stateful Session beans.

Slide 14

Let us understand the features of EJB 3.0 in depth.

EJB 3.0

- ❑ Some of the important features introduced in EJB 3.0 are as follows:
 - Use of annotations
 - Callback Methods
 - Elimination of Home interface
 - Elimination of component interface
 - Dependency Injection
 - Interceptors
 - Java Persistence API (JPA)
 - Timer service

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 14

Use slide 14 to introduce the features specific to EJB 3.0.

Annotations are metadata which are introduced in EJB 3.0. They enable the developer to define the required resources and deployment information within the application code. They are used in addition to the deployment descriptors. However, if same feature is defined both through annotations and deployment descriptors, then the definition in deployment descriptor has higher priority than the annotations.

Callback methods are those methods which are invoked when various lifecycle events with respect to the enterprise bean occur in the application. These methods are defined through predefined interfaces; the developer has to write the code as per requirement.

Earlier versions used home interface and remote interface that was accessed by application clients. Home interface was replaced by the business interface in EJB 3.0.

Component interface in earlier versions were used as a way for the container to notify the bean instance of various lifecycle events affecting it. This has been removed from EJB 3.0. The lifecycle callback methods of Stateful Session beans do the needful in the current version of EJB.

Dependency injection is introduced in EJB 3.0 to keep the application lighter. Required resources are instantiated into the application only if required. This is possible through the annotations which indicate the resources required for the bean class.

Interceptors are methods used to keep track of method invocation, invocation of business methods, and so on. These interceptors are executed either before or after the method execution to keep track of various aspects of method execution.

Java Persistence API handles all the database operations and replaces the utility of Entity beans of earlier versions.

Timer service is introduced in EJB 3.0. It enables developers to schedule certain application tasks based on calendar expressions without explicit intervention.

Slide 15

Let us understand the usage of annotations.

Use of Annotations 1-2

- ❑ EJB 3.0 uses metadata annotations to specify the services used by the EJB components.
- ❑ Metadata annotations simplify the development and testing of the application.
- ❑ Annotations:
 - Enable developer to provide the specification based on which code is added.
 - Processed at compile time.

- Following code snippet shows the use of annotations:

```
...  
@Stateful  
public class HelloBean  
implements  
HelloInterface {  
  
@Remove  
public void removeBean()  
{  
//close all resources  
}  
}  
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 15

Use slide 15 to explain how annotations are used in applications for different purposes.

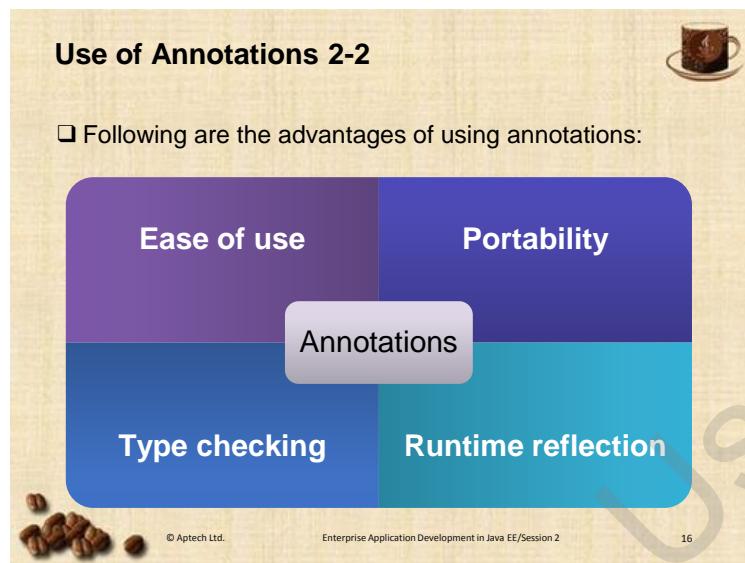
Annotations are metadata used to give information to the application tester and deployer. They are also used to specify required resources for the application.

Annotations help the developer to provide the specifications and based on that specifications, the system automatically adds code. Metadata annotations simplify the development and testing of the application. Annotations, which are used extensively in the Java enterprise platform that help the developers to transform a simple POJO to an EJB. Besides specifying the required services, annotations can also be used to specify the component type.

Annotations are processed at compile time. They are also used during application deployment.

Slide 16

Let us understand the advantages of annotations.



Use slide 16 to explain the advantages of annotations.

Some of the advantages of using annotations are as follows:

Ease of use – Annotations are checked and compiled by the Java language compiler and are simple to use. Many vendors such as IBM and BEA have introduced the annotation feature in attributes for the deployment descriptor.

Portability – Annotations are portable.

Type Checking – Annotations are instances of annotation types and are compiled in their own class files.

Runtime Reflection – Annotations are stored in the .class files and accessed for runtime access.

Slide 17

Let us understand dependency injection.

Dependency Injection

- ❑ Means through which the container creates the required operational environment for the application.
- ❑ Required resources are injected based on annotations and deployment descriptors.
- ❑ Annotations used are:
 - `@Inject` and `@Resource` annotations.
- ❑ Dependency injection is also supported through JNDI lookup.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 17

Use slide 17 to introduce dependency injection in enterprise applications.

In an enterprise application, if all the resources are instantiated at deployment, the application would occupy a lot of memory and block application resources. This strategy reduces application efficiency. Dependency injection is used to improve the application efficiency by injecting only the required resources into the application.

Java EE implements resource injection and dependency injection through annotations. `@Inject` and `@Resource` annotations are used for this purpose.

When the application requires resource injection or dependency injection it uses the services of JNDI to lookup resource and instantiate into the application. The `lookup()` used for JNDI lookup is a member of `EJBContext` interface.

Slide 18

Let us understand callback methods.

Callback Methods

- ❑ Callback methods are those methods which are invoked when various lifecycle events with respect to the enterprise bean occur in the application.
- ❑ Following are different callback methods defined in EJB 3.0:

ejbActivate	ejbPassivate
ejbLoad	ejbStore
- ❑ Another change introduced in EJB 3.0 is that any method can be designated as callback method to listen to lifecycle events.

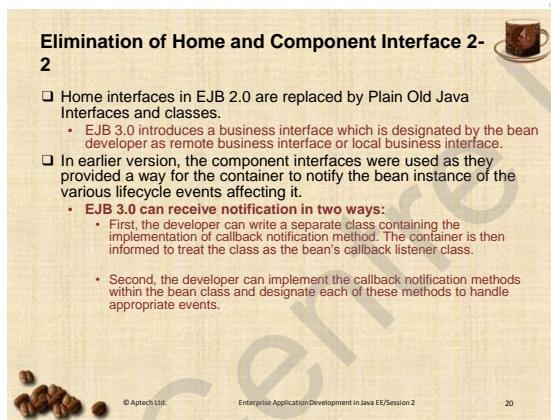
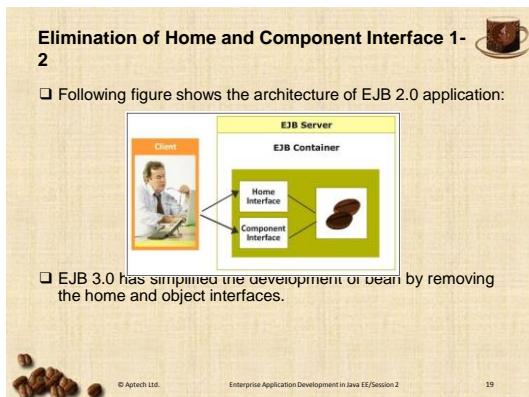
© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 18

Use slide 18 to explain the purpose of callback methods in enterprise applications. Application components are instantiated and deployed into the container during application execution. Application may require additional operations to be performed during this instantiation such as logging the instantiation, keeping track of memory used, and so on. Callback methods perform these operations.

EJB 3.0 provides callback methods such as `ejbActivate()`, `ejbPassivate()`, `ejbLoad()`, and `ejbStore()` methods. These methods are used with session beans. Apart from these, methods developers can define their own callback methods in the application. These callback methods have to be annotated to indicate the time when they are expected to be invoked.

Slides 19 and 20

Let us understand how Home interface and Component interface are removed from EJB 3.0.



Use slides 19 and 20 to explain what are Home interface and Component interface. These aspects are removed from EJB 3.0. Explain how removal of these concepts from EJB 3.0 has improved the performance of EJB 3.0.

The Home interface provides bean management and lifecycle methods. It specifies methods that allow the client to create, remove, and find objects of the same type. It also provides definitions for home business methods for entity beans. Home business methods are not specific to a particular bean instance, they are applied to all the instances of a bean class. Home interface is defined by the developer and it allows the client to create bean objects.

Enterprise Java Bean functionality is accessed by means of the bean's component interface, which defines the business methods visible to and callable by the client. It does not allow creating and manipulating the bean objects but the bean functionality can be accessed through component interface. These component interfaces contain the declaration of various lifecycle methods such as ejbPassivate, ejbActivate, ejbLoad, and so on. Earlier, it was mandatory to implement these methods in the bean class. EJB 3.0 onwards the developer need not implement the lifecycle callback methods of the beans.

EJB 3.0 did away with these two interfaces and only provides a business interface. The business interface is equivalent to the Component interface in earlier versions; the client cannot instantiate the beans in EJB 3.0. This task is carried out by the container in EJB 3.0. The container instantiates and creates a pool of beans which can be accessed by the client applications.

Slide 21

Let us understand Interceptors.

Interceptors

- ❑ Are the methods used to intercept business method calls or lifecycle callback method calls.
- ❑ Can be used by Stateless, Stateful, and Message-driven beans.
- ❑ Perform operations such as application auditing, logging, and so on.
- ❑ Are defined as methods in the bean class or as a separate interceptor class in the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 21

Use slide 21 to introduce interceptors to the students.

EJB 2.0 provides in-built Interceptor support such as transactions and security through deployment descriptor by specifying security and transactions details at method level. Earlier versions were not providing interceptor support to business method invocation, in EJB 3.0 interceptors can be defined for business methods. Method level interceptors and class level interceptors can be defined in EJB 3.0.

Interceptors can be used by Stateless, Stateful, and Message-driven beans.

Slide 22

Let us understand Java Persistence API.

Java Persistence API

- Is the persistence technology used to persist enterprise beans in the databases.
- Bridges the gap between object-oriented interpretation in Java programs and relational database.
- Enables writing code independent of underlying database provider.
- Provides an enhanced EJB query language which supports execution of EJB queries.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 22

Use slide 22 to introduce Java Persistence API to the students.

Earlier versions of EJB used Entity beans to implement database operations. Persistence API provides EJB query language which is similar to structured query language used in database systems.

Java Persistence API enables object-oriented interpretation of data in the application. It also enables developers to write object-oriented code independent of underlying database. It provides a method of managing data between the object-oriented Java application and relational database.

In-Class Question:

After you finish explaining Java Persistence API, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the function of JPA API?

Answer:

Relational databases interpret data in the form of tables, where each table has a set of rows and columns. Object-oriented programming interprets data in the form of objects with properties and behavior. The API has to bridge this gap of data interpretation between the two technologies.

Slide 23

Let us understand Timer Service.

Timer Service

- ❑ Timer service is used to schedule application at specific time intervals.
- ❑ Timer service enables the developer to schedule various events of the application without manual intervention.
- ❑ Developers can set up timers and define timer callback methods in the application which are invoked when the timer expires.

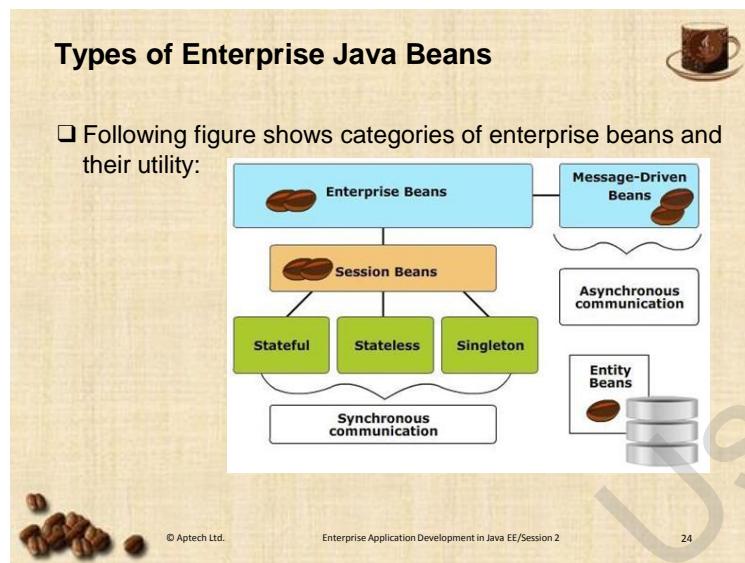
© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 23

Use slide 23 to introduce Timer Service which is a feature introduced in EJB 3.0.

Timer service enables developers to schedule application tasks without intervention. Developers can schedule tasks based on absolute calendar expressions or relative time expressions.

Slide 24

Let us understand the different categories of enterprise beans.



Use slide 24 to explain the different categories of enterprise beans provided by EJB 3.0 specification.

The image in the slide shows three categories of beans, Session beans, Message-driven beans, and Entity beans.

Session beans are used to manage communication between the client and the server. This communication is termed as session hence the name Session bean. There are three variants of Session beans Stateless session beans, Stateful session beans, and Singleton session bean.

Stateless session beans communicate with the application clients, but do not save the communication state to permanent storage.

Stateful session beans communicate with the application client and store the communication onto permanent storage after the communication session ends. When the client accesses the application again the stored state is retrieved and used for current communication session.

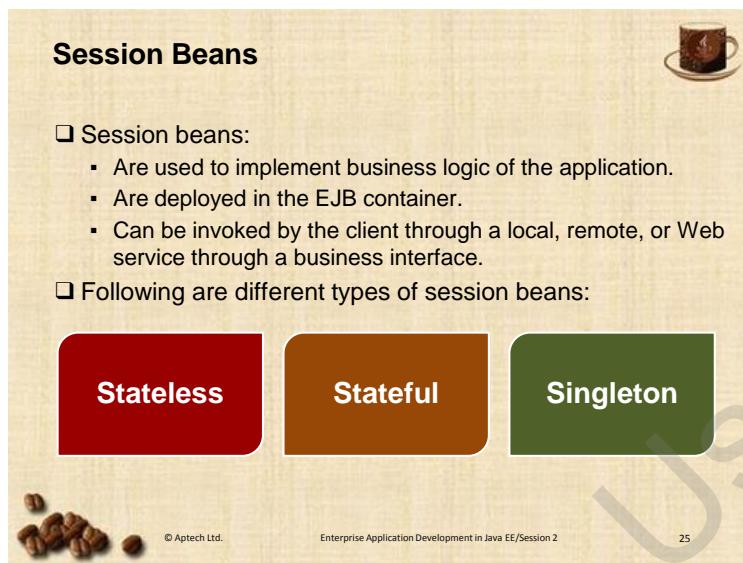
Singleton session bean is instantiated only once during application lifecycle and is retained for the entire application lifecycle.

EJB 3.0 also supports Message-Driven beans. EJB3.0 supports asynchronous communication among the application components through message exchange. When a message is exchanged between the application components it has to be processed. Message-driven beans are used to process these asynchronous messages.

Earlier versions of EJB managed database operations of the application through Entity beans. EJB 3.0 has JPA API to implement database operations, however, it has to support the database operations performed in earlier versions, hence it also supports Entity beans.

Slide 25

Let us understand Session beans.



The slide has a light beige background with a faint watermark of a coffee cup and coffee beans. At the top left, the title "Session Beans" is displayed in bold black font. In the top right corner, there is a small graphic of a coffee cup. At the bottom left, there is a small graphic of coffee beans. The slide contains the following text and buttons:

- ❑ Session beans:
 - Are used to implement business logic of the application.
 - Are deployed in the EJB container.
 - Can be invoked by the client through a local, remote, or Web service through a business interface.
- ❑ Following are different types of session beans:

Three buttons are present below the text:
Stateless (red button)
Stateful (brown button)
Singleton (green button)

Small text at the bottom center: © Aptech Ltd.
Enterprise Application Development in Java EE/Session 2
25

Use slide 25 to introduce Session beans. Tell the students that this is the basic introduction of Session beans.

Session beans implement the interaction with application clients and processes the client requests. They are server-side components which are deployed in containers. The container receives the client request and forwards it to appropriate EJB in the container.

An EJB can be accessed through one of the three interfaces, no-interface view, local interface, and remote interface. The developer can define all the three interfaces or limit it to any one of the three interfaces based on the application requirement.

Slide 26

Let us understand Message-driven beans.

Message-Driven Beans

- Enterprise beans asynchronously invoked through Java Message Service(JMS) messages.
- Messages can be sent from the application client or another application component.
- Message-driven beans cannot be invoked through interfaces.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 26

Use slide 26 to introduce Message-driven beans to the students. Java EE enables asynchronous communication among application components through message exchange.

Java EE provides JMS API which enables defining messages for asynchronous communication and processing of the messages.

Message-driven beans are invoked by these JMS messages. They cannot be explicitly invoked by the application client or through business interfaces. Give an example suitable to the class profile to explain why an application should have JMS messages.

Slide 27

Let us understand Entity beans.

Entity Beans

- ❑ There data gets persisted in the database storage.
- ❑ They also implements business logic pertaining to the data in the application database.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 27

Use slide 27 to briefly explain Entity beans. Remember Entity beans are not a feature of EJB 3.0. EJB 3.0 introduces Java Persistence API to implement data persistence in the application. It implements a plain POJO class which is persisted to the database.

Entity bean is an object whose data gets persisted in the database storage. It also implements business logic pertaining to the data in the application database. It manages application data and returns required data from the database. It is a server-side component.

Slides 28 to 30

Let us understand container services.

Container Services for EJB 1-3



- ❑ According to the security model implemented by a Java EE container, the container authenticates and authorizes users wanting to access the application and system resources.
- ❑ The transaction model of the container ensures that the methods in a single transaction are appropriately executed without leading to an inconsistent application state.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 28

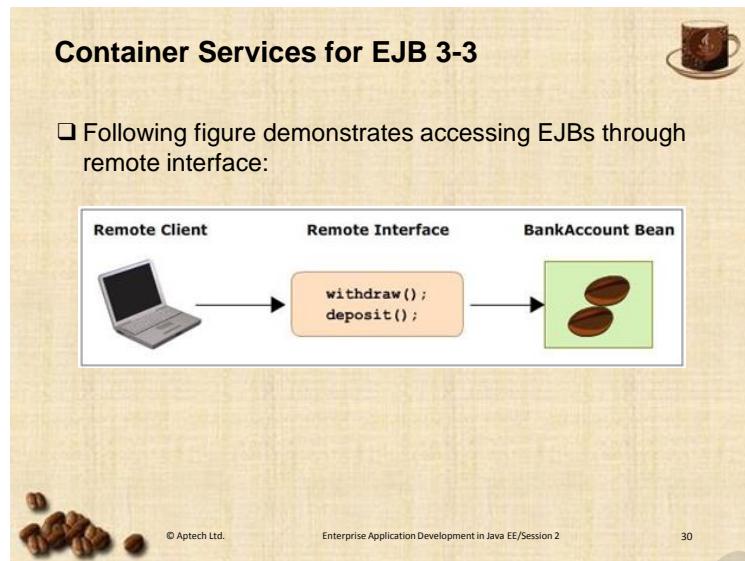
Container Services for EJB 2-3



- ❑ The container provides naming services to access different components of the application without any conflict.
- ❑ The container also provides connectivity services to the application. When an application component deployed in the container attempts to connect to an external component, then the container manages all the lower-level communication tasks.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 29



Use slides 28 to 30 to explain the role of containers in enterprise applications.

Containers provide generic services to the enterprise applications. All the client requests for application service first reach the container; the container in turn invokes the EJB. Thus, the client cannot directly access the EJB, the container protects the EJB from direct external access.

Apart from avoiding direct access, the container can also run authentication and authorization processes to protect the application components. Before servicing the request, the container can ask for username and password from the client and if the user is allowed access then only the container provides access to the clients.

While processing the client requests the EJB may require access to the database. The EJB initiates a transaction to access database. However, this request from the EJB reaches the container and container then manages the entire transaction. If there is any intermediate failures the container will reinitiate and complete the transaction, the EJB will be unaware of these intermediate failures.

All the application components have to be uniformly accessed in the application. Java EE provides a naming scheme for all the application components through JNDI. When an application component is trying to access another application component, the container locates the component based on its JNDI name and provides access to the requesting component.

As enterprise applications are developed based on distributed architecture, the application components might be located on different locations. The container is responsible for communication among the application components and enables efficient functioning of the application.

The figure given on slide 30 demonstrates how a client can access the BankAccount Bean deployed on application server through a remote interface. Tell the students that all remote clients can access the EJBs only through remote interface. The remote interface has methods `withdraw()` and `deposit()` which can be invoked by the remote client.

Slides 31 and 32

Let us understand how clients can access EJBs.

Accessing Enterprise Java Beans 1-2



Enterprise beans are accessed through the following interfaces:

- Business interface refers to the set of methods provided by the bean class through which the enterprise bean can be invoked.
- No-interface view refers to all the public methods of the bean class which can be used to access the bean.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 31

Accessing Enterprise Java Beans 2-2



Following code snippet demonstrates the business interface of **Calculator** bean:

```
package Test;
import javax.ejb.Remote;
@Remote
public interface Calculator {
    public String sayHello(String name);
    public int addition(int a,int b);
    int multiplication(int a,int b);
    int subtraction(int a,int b);
}
```

The interface has four methods `sayHello()`, `addition()`, `multiplication()`, and `subtraction()`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 32

Use slides 31 and 32 to explain the interfaces through which application clients can access the EJBs. EJBs can be accessed by clients through interfaces. There are two types of clients which can access the enterprise application, local clients, and remote clients.

Local clients can access the EJB either through no-interface view or through business interface. Remote clients can access the EJB through remote interface. An interface of an EJB is the methods of the EJB which can be invoked.

No-interface view is used by local clients to access the EJB. It comprises all the public methods of the EJB. Local clients can invoke these methods.

Business interface comprises all the business methods exposed to the client. This can be used by both the local clients and remote clients as defined by the developer. If the business interface is annotated with `@Remote` then it implies that it can also be accessed by remote interface.

Developers can define multiple interfaces for different categories of clients.

Slide 32 shows an example interface for a **CalculatorBean()**, the interface implies all the operations which can be performed by the bean.

Slide 33

Let us understand local clients of an enterprise application.

Local Clients

- ❑ A local client runs in the same application where enterprise bean is accessing.
- ❑ A local client can either be a bean component or a Web component.
- ❑ A local client can access the enterprise bean through:
 - No-interface view
 - Business interface annotated with @Local

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 33

Use slide 33 to explain what local clients in an enterprise application are. A local client runs on the same Java Virtual Machine (JVM) as the enterprise bean. A local client can be another bean component or Web component or an end user.

Local clients access the EJB either through no-interface view or local interface. Local clients require JNDI lookup to access the EJB in the application. Local client can be another component of the same application or an end user.

Slide 34

Let us understand remote clients.

Remote Clients

- ❑ Remote client can run on a different machine, different JVM or application.
- ❑ Remote client can be a bean component, Web component, or application client.
- ❑ Remote clients can access the enterprise bean through a remote interface.
- ❑ Remote interface should be annotated with `@Remote`.
- ❑ Remote clients cannot access enterprise bean through no-interface view.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 34

Use slide 34 to explain remote clients. Remote clients are application components which run on a different virtual machine than the EJB. Remote clients can also access the EJB over network.

A remote interface is annotated with `@Remote` annotation. A remote client cannot access an EJB through no-interface view.

Tips:

Remote interfaces can be looked up by using `<ejb-ref>` element in the deployment descriptor of the application.

```
<ejb-ref>
    <ejb-ref-name>ejb/EJBTest</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local>EJBTest</local>
</ejb-ref>
```

Slides 35 to 43

Let us understand Java Naming and Directory Interface (JNDI).

**Java Naming and Directory Interface
1-9**



For Aptech Only

- ❑ A large scale enterprise domain has large number of objects interacting with each other.
- ❑ These applications require a well-defined naming infrastructure to access objects as per the requirement.
- ❑ JNDI is an API which provides these naming services.
- ❑ JNDI binds a name with an object in the application.

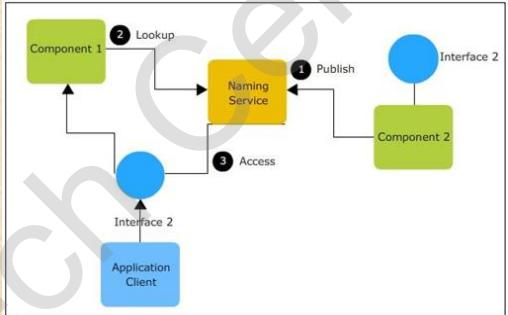
© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 35

**Java Naming and Directory Interface
2-9**



For Aptech Only

- ❑ Following figure demonstrates how clients can access objects through a naming service:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 36

Java Naming and Directory Interface 3-9



- All application components publish their identity to the naming service.
- When an application component has to access a bean it performs a JNDI lookup.
- The naming service then provides reference to the application component.
- All remote clients of the application access the object through the JNDI name.
- JNDI organizes the objects in the namespace in a hierarchy.



© Aptech Ltd.

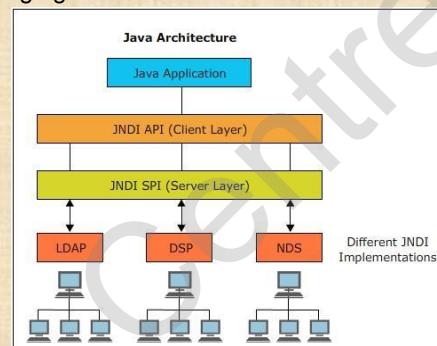
Enterprise Application Development in Java EE/Session 2

37

Java Naming and Directory Interface 4-9



- Following figure shows the JNDI architecture:



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 2

38

Java Naming and Directory Interface 5-9



- The JNDI architecture contains two parts:
 - **JNDI API** – Are used by the remote applications to access the mapped components or objects from JNDI registry.
 - **JNDI Service Provider Interface (SPI)** – Is a mechanism to plugin naming and directory services of different vendors on the Java EE platform.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 2

39

Java Naming and Directory Interface

6-9

❑ `java.naming` is the main package of JNDI API.

❑ `java.naming` has the following important classes:

- `InitialContext` – used to establish connection and retrieve `Context` object.
- `Context` – provides methods for binding and unbinding JNDI names with objects.

❑ Following code snippet shows how to retrieve the `Context` object:

```
Context ctx = new InitialContext();
```

Java Naming and Directory Interface

7-9

❑ Following are the methods which are part of `Context` class are:

- `bind()`
- `unbind()`
- `lookup()`
- `rebind()`

Java Naming and Directory Interface

8-9

❑ The given figure demonstrates how to obtain a JNDI context using `InitialContext()`:

- The client performs the lookup of the objects registered in the JNDI registry.
- The application server provides the services for binding and unbinding the objects to the JNDI registry.

Java Naming and Directory Interface

9-9

- JNDI provides three namespaces:
 - java:global identifies an object through a hierarchy of application name
 - Format: `java:global[/application name]/module name /enterprise bean name[/interface name]`
 - java:module identifies the component from module level of naming hierarchy
 - Format: `java:module/enterprise bean name/[interface name]`
 - java:app identifies the enterprise beans packaged within the application
 - Format: `java:app[/module name]/enterprise bean name [/interface name]`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 43

Use slides 35 to 43 to explain the use of JNDI in an enterprise application.

An enterprise application has multiple components which are to be accessed during application execution. There must be a uniform convention to access all the application components remotely by the clients. Java EE provides a service named Java Naming and Directory Interface (JNDI) for registering the components of the enterprise applications.

JNDI names of all the components are arranged in a hierarchy and each name refers to a unique application component. The reference associated with each component is used to access the application component by the client.

JNDI is an API which provides these naming services. JNDI binds a name with an object in the application.

Use slides 36 and 37 to explain how clients access the objects through a naming service. There are three major operations associated with JNDI service – Publish, Lookup, and Access.

The figure as shown on slide 36 explains how a component 2 is publishing its identity to the naming service. When component 1 tries to access component 2 it will perform a lookup operation onto the naming service. An application client which is trying to access the component 2 will access it through the interface provided.

All the application components publish their identity to the naming service. Naming service binds a component with a JNDI name. When an application client or component has to access a bean it has to first perform a lookup based on the JNDI name. The naming service then provides reference of the component.

The client refers to an object. Through an object, the naming service implemented by JNDI is responsible for accessing the object using appropriate object references. All the remote clients of the application access the object through JNDI name. JNDI organizes the objects in the namespace in a hierarchy. The resource objects which can be accessed through JNDI names are enterprise beans, databases, and so on.

Use slide 38 to explain the JNDI architecture. JNDI architecture has a Service Provider Interface (SPI) and a JNDI API. The JNDI API is used by enterprise applications to use the naming service.

The Service Provider Interface allows different naming services to be plugged in transparently to the JNDI API. There are several implementations of naming services such as Lightweight Directory Access Protocol (LDAP), Directory System Protocol (DSP), and Netware Directory Services (NDS). These implementations of naming service can be used in an enterprise application with the help of the SPI.

Use slide 39 to explain the functions of JNDI API and JNDI SPI.

Tell them that an application access the JNDI service using JNDI API which provides a standard way for accessing the underlying objects or directory service.

The JNDI architecture contains two parts:

- **JNDI API** – Are used by the remote applications to access the mapped components or objects from JNDI registry.
- **JNDI Service Provider Interface (SPI)** – Is a mechanism to plugin naming and directory services of different vendors on the Java EE platform.

Then, use slide 40 to explain the structure of JNDI API.

Tell them that JNDI API contains interfaces namely, Context and Name. It also contains a class named `InitialContext` which represents the root node of JNDI hierarchy.

`java.naming` package is the main package of the API. The `java.naming` contains classes and interfaces to access naming services.

The `InitialContext` is an important classes of `javax.naming` package. The `InitialContext` provides a context for the naming service hierarchy. It provides a starting point for naming and directory operations. It is used as a reference for all the naming and directory operations.

The `Context` is the core interface of the `javax.naming` package, it provides operations for looking up, binding/unbinding of JNDI names, creating, and destroying application contexts.

Tips:

JNDI has five different packages – `javax.naming`, `javax.naming.directory`, `javax.naming.ldap`, `javax.naming.event`, `javax.naming.spi`.

`javax.naming.directory` package provides functionality for accessing directory services along with naming services.

`javax.naming.ldap` package has classes with features specific to LDAP.

`javax.naming.event` package contains classes and interfaces to support event notification in naming and directory services.

`javax.naming.spi` package provides classes and interfaces which enable various implementations of naming services to be plugged into the JNDI API.

Then, explain them that the `InitialContext` class implements the `Context` interface and provides the root of the hierarchy, relative to which all the objects in the namespace are named.

Firstly, client establishes a connection to the JNDI service, also referred to as JNDI tree. After getting the connection to the JNDI service running on the Java EE platform, an object of `Context` is created. The `Context` object allows access to the system components and resources. The context representing a naming context consists of name-to-object bindings.

The client uses the `InitialContext` to establish the connection and retrieving the `Context` object. For example, to retrieve the `Context` object, the following statement is used by the client:

```
Context ctx = new InitialContext();
```

The reference to the `Context` provides methods for binding and unbinding JNDI names with objects. It also allows creating sub contexts and provides methods for creating sub contexts.

Use slide 41 to explain various methods provided by the `Context` interface.

- `bind()` - It accepts two parameters, the JNDI name in string format and the object to which it has to be bound.
- `unbind()` - It accepts a string parameter and is used to unbind the specific object from the JNDI name.
- `lookup()` - It returns an object on which the lookup operation is performed. It accepts the JNDI name as parameter.
- `rebind()` - It also accepts two parameters, the JNDI name and the object.

Use slide 42 to explain the process of obtaining JNDI context using `InitialContext`. An `InitialContext` of the application has to be obtained before performing any operation on naming or directory service. The `InitialContext` represents the starting point of the namespace. JNDI organizes all the objects in the namespace in a hierarchy, the `InitialContext` provides the root of the hierarchy.

The client performs the lookup of the objects registered in the JNDI registry and the application server provides the services for binding and unbinding the objects to the JNDI registry.

Use slide 43 to explain different naming schemes of accessing Java objects in the application. JNDI supports three types of naming schemes – global namespace, module namespace, and local namespace.

The global namespace identifies the application component through the application name, module name, and enterprise bean name. Following is the format of `java:global` namespace:

```
java:global[/application name]/module name /enterprise bean  
name[/interface name ]
```

The module namespace identifies the EJB component within the application context. The naming URL is relative to the application name.

The format of the `java:module` name space is given as follows:

`java:module/enterprise bean name/[interface name]`

The app namespace identifies the EJB component within the context of the module which is in turn in the context of the application.

The format of the `java:app` namespace is as shown:

`java:app[/module name]/enterprise bean name [/interface name]`

In-Class Question:

After you finish explaining the use of JNDI in enterprise applications, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which of the following represents the root node of JNDI hierarchy?

Answer:

`InitialContext`

Additional References:

Prepare with an example to be discussed on JNDI, to do so, visit the following link:

http://www.tutorialspoint.com/ejb/ejb_jndi_bindings.htm

To get more information on JNDI, visit the following link:

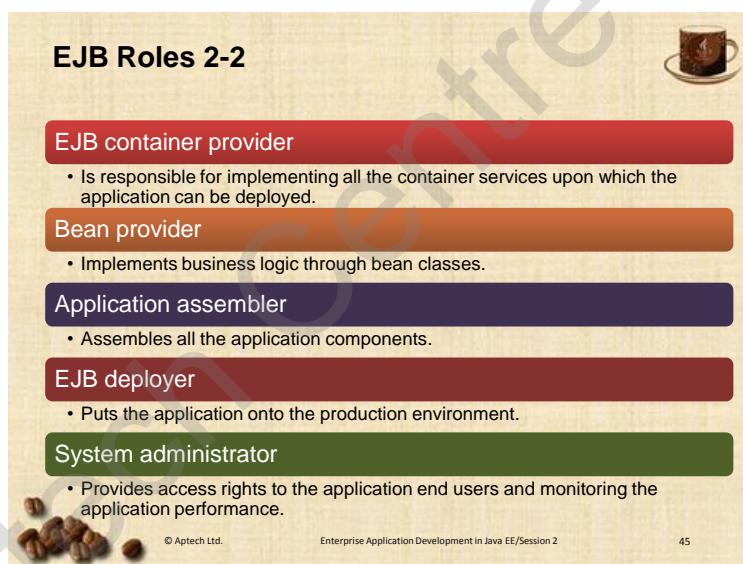
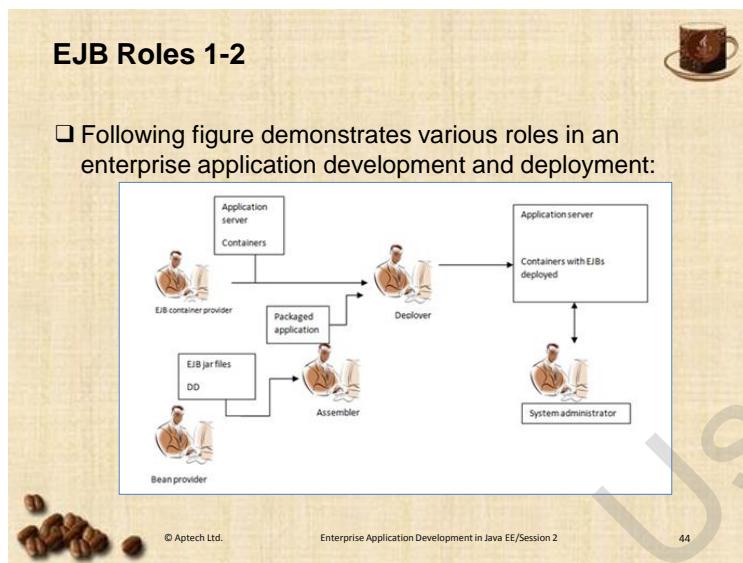
<http://www.javaworld.com/article/2076888/core-java/jndi-overview--part-1--an-introduction-to-naming-services.html>

To get more information on JNDI namespace and their scope, visit the following link:

<http://www.javacodegeeks.com/2011/08/ejb-31-global-jndi-access.html>

Slides 44 and 45

Let us understand different roles associated with enterprise application development.



Use slides 44 and 45 to explain different roles in enterprise application development and deployment.

Tell them that an application development process has various stages such as design, development, deployment, and administration. The design phase involves identifying the classes and objects that should be implemented for the application. It also defines the interactions among the objects of the domain. The development stage involves the coding to implement the components using different technologies. The components developed in the development stage are deployed on the application server. Once the application is deployed, the access to the application should be given to different users of the application based on their roles and responsibilities in the application domain.

At every stage of application development there is a task to be implemented which is carried out by different roles assigned.

Then, explain various roles in the application and their mutual interaction in application development and deployment.

The developed application components are packaged as Java archive files. There are various roles in application deployment.

Bean provider provides the implementation of EJB 3.0 specification. This serves as a platform for application deployment.

EJB container provider provides the implementation of EJB container for the application server.

The application assembler receives all the packaged components of the application, the assembler integrates the application.

The application deployer uses the container implementation provided by EJB container provider and deploys the assembled application on the container. Deployer is responsible for making the application operational in real time environment.

System administrator is responsible for managing the application, defining who can access the application, and so on. The administrator is also responsible for monitoring the application performance and informing other stakeholders if there are performance related issues.

In-Class Question:

After you finish explaining different roles in enterprise application development and deployment, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



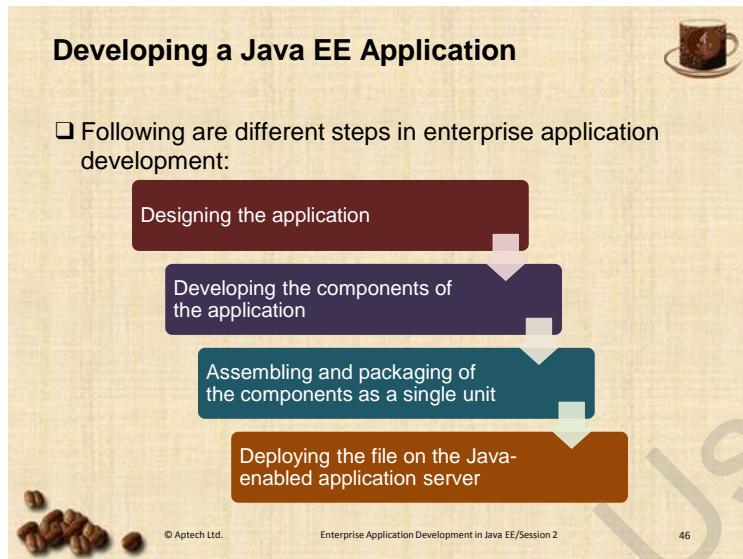
Who is responsible for securing the application through a firewall and makes hardware choices for appropriate application deployment?

Answer:

EJB Deployer

Slide 46

Let us understand the process of developing a Java EE application.



Use slide 46 to explain the process of developing an enterprise application.

Based on the application requirements the application, designers create design documents which comprises class diagrams, use case diagrams, and other tools which can be used in the creation of an object-oriented enterprise application.

Java enterprise applications are developed based on the component model of the application. Different components of the application are independently developed and integrated into a single package. During the integration phase additional code might be written to make the application components compatible.

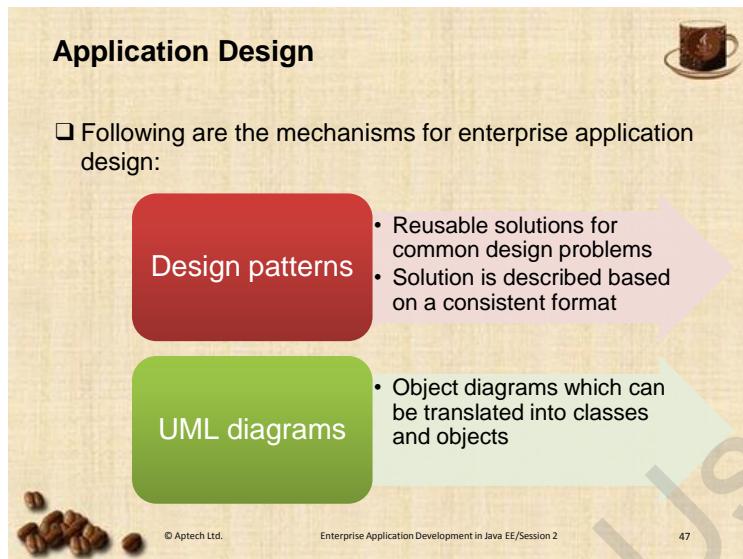
Finally the assembled application is deployed on the application infrastructure.

Tips:

The process of developing an enterprise application starts by collecting the application requirements from the end users. This phase is also known as **Requirement analysis phase**.

Slide 47

Let us understand the application design process.



Use slide 47 to explain the tools used in the application design process. Design patterns and UML diagrams are two essential tools in the Java enterprise application development.

Design patterns are templates which can be used by application developers. The templates characterize a recurring problem and provide a tried and tested solution for the problem. This reduces the problem solving time if a similar problem has been solved earlier and documented.

UML stands for Unified Modeling language. It is a graphical language used to model the application domain with the help of class diagrams, use case diagrams, and other diagrams. This representation can be used by the application developers to create classes and respective objects.

Tips:

UML diagrams are used in the designing of object-oriented applications.

Slide 48

Let us understand the coding phase of enterprise application development.

Coding

- ❑ Involves writing code for various application components.
- ❑ Application components include user interface, database, enterprise beans, and so on.
- ❑ Application is developed using NetBeans IDE.
- ❑ Developers develop application components, debug, and perform unit testing during this phase of application development.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 48

Use slide 48 to explain the process of coding during enterprise application development. Developers create application components during this phase. Developers use design documents to create classes, user interface of the application, create database component, and so on.

Developers use Integrated Development Environment (IDE) for application development. IDEs simplify the process of creating different application components and integrating them. IDEs also enable application debugging and testing different components.

Tips:

NetBeans and Eclipse are popularly used Java compatible IDEs.

Slides 49 and 50

Let us understand the role of deployment descriptors.

Deployment Descriptor 1-2



❑ Describes interaction between various components of the application in declarative fashion.
 ❑ Is an XML file used by Java-enabled application server to obtain the references of the components.
 ❑ Includes the following information:

- Transaction management policy of the application.
- Security specifications and authorization constraints.
- Configuration variables used during application development.
- Resource access dependencies of the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 49

Deployment Descriptor 2-2



❑ Following code snippet shows the code of the deployment descriptor `glassfish-ejb-jar.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC . . .>
<glassfish-ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>CalculatorImplBean</ejb-name>
            <jndi-name>caljndi</jndi-name>
            <business-local>CalculatorLocal</business-local>
            <business-remote>CalculatorRemote</business-remote>
            <ejb-class>beans.CalculatorImpl</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
</glassfish-ejb-jar>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 50

Use slides 49 and 50 to explain the role of deployment descriptor in enterprise applications. Deployment descriptors are declarative .xml files which are used for application deployment. It has all the configuration details for application deployment on the application server. It also has security information and transaction management information in it.

Every enterprise application has at least one deployment descriptor. Applications can have more than one deployment descriptors.

The `ejb-jar.xml` is standard deployment descriptor for EJB based applications and `web.xml` is a standard deployment descriptor for Web based applications.

Use slide 50 to show a sample deployment descriptor. The deployment descriptor is an .xml file named `glassfish-ejb-jar.xml`.

This implies that it is a deployment descriptor for deploying the application on Glassfish server. The file has deployment information for the EJB `CalculatorImplBean`. The JNDI name of the EJB is `caljndi`. According to the deployment descriptor the EJB has a local interface and remote interface named, `CalculatorLocal` and `CalculatorRemote` respectively. The EJB class for the `CalculatorImplBean` is located in the directory `beans` and the class is `CalculatorImpl`.

Tips:

EJB 3.0 uses annotations in addition to deployment descriptor to provide configuration information and security information. In case certain configuration is given through deployment descriptor and annotation then deployment descriptor is given preference.

Slide 51

Let us understand application packaging.

Packaging 1-2

□ The steps for packaging and deploying a Java EE application are as follows:

- The Application Component Providers create Java EE modules, such as EJB, Web, Resource Adapter, and Application clients.
- These modules can be deployed independently without being packaged into a Java EE enterprise application.
- The Application Assembler packages these modules to create a complete Java EE enterprise application.
- Deployer deploys the deployable unit.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 51

Use slide 51 to explain different aspects of application packaging.

An application is packaged into a deployable unit after development. The application can be packaged into an archive file or the components can be independently deployed.

Independently deployed components are assembled by the assembler. If there are any dependencies among application components the application assembler must resolve these dependencies before deploying the application. The deployer then deploys these independent modules onto the application server.

Tips:

An archive file is a self-contained unit which has the configuring information and all the components. Based on the information provided in the deployment descriptor the application can be deployed on the application server. It may contain all the required resource adapters and drivers for proper application functioning. Enterprise applications are archived as enterprise archive files (EAR files) and Web applications are packaged as Web Archive files (WAR files).

Slide 52

Let us understand how independent modules can be packaged into archive files.

Packaging 2-2

- The Java EE specification provides different types of archive files to package the modules. These are as follows:
 - EJB modules**
 - Comprise all class files
 - Class files are packaged as .jar files
 - Web modules**
 - Comprise all HTML files, JSPs, JSFs, and Servlets
 - These files are packaged as .war files
 - Resource adapter modules**
 - Components for resource adaptation
 - Packages a .rar files
 - Application client modules**
 - These are class files packaged as .jar files

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 52

Use slide 52 to explain how different modules can be packaged into different archive files.

- **EJB modules** comprise all the class files pertaining to the business logic of the application. EJB modules are packaged as .jar files and may have a deployment descriptor.
- **Web modules** comprises the HTML files, JSPs, JSFs, and Servlet class files pertaining to the application. All the Web application files are packaged into a .war file for deployment and distribution. The archive file of the application may or may not have the deployment descriptor.
- **Resource adapter modules** comprise components which implement resource adaptation. In scenarios where the application is communicating with a database, resource adapter objects play an important role to translate the database interpretation of data to object-oriented interpretation. It also includes the native libraries used by the application. The resource adapter files are packaged as archive files using .rar extension.
- **Application client modules** are modules like user forms for the application; these modules are packaged as .jar files.

Slides 53 to 56

Let us understand the application deployment process.

Assembling and Deployment 1-4



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 53

- ❑ Independent components of the application are to be assembled together.
- ❑ Assembled application has to be deployed on the application server.
- ❑ Application is deployed either manually or through a tool.
- ❑ Deployment process also involves vendor specific configuration with respect to load balancing and performance tuning of application components.

Assembling and Deployment 2-4



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 54

- ❑ The Application Assembler packages the components into an Enterprise Archive (EAR) file.
- ❑ The assembler should ensure the root contains a folder called META-INF.
 - This folder contains a deployment descriptor called application.xml, one or more container-specific deployment descriptors, and manifest.mf file.
 - The application.xml file contains the names of all the Java EE archives that are packaged in the EAR file.
 - The manifest.mf file contains additional meta-information about the EAR file.

Assembling and Deployment 3-4



Besides the META-INF folder, the root of the file structure contains the archives of the Java EE modules that constitute an EAR file.

These archives can be:

- .jar files for EJB module
- .war files for Web modules
- .jar files for Application Clients
- .rar files for Resource Adapters

These archives can be either placed in the root or in one or more sub-folders.

To manually package the components in an EAR file, the command can be run as:

```
* jar cvf <name>.ear *
```

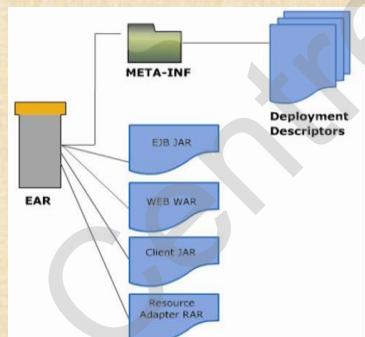


© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 55

Assembling and Deployment 4-4



Following figure shows the file structure of EAR file:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 56

Use slides 53 to 56 to explain the application deployment process of Java EE enterprise applications.

Application requires system specific folders for proper application deployment. Every application has at least one deployment descriptor. A packaged application has a metadata folder which is used for application deployment.

Each of the application components is packaged into respective archive files. The container specific deployment information is present in `manifest.xml` file. The `META-INF` folder of the application archive files has all the application modules and their respective configuration information. The archive files of the modules can be placed in sub folders of `META-INF` directory or in the directory.

Slides 57 and 58

Let us understand the IDEs used for Java EE application development.

EJB Development Tools 1-2



The EJB development environment is provided by Integrated Development Environments (IDEs) such as NetBeans, Eclipse, and so on.

Following are different categories of tools provided by the development environment:

- J2EE Perspective editor
- Tools for creating and accessing enterprise beans
- Tools for accessing the database
- Tools for generating deployment code from the annotations
- Tools for creating applications based on design patterns

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 57

EJB Development Tools 2-2



Following figure shows the NetBeans IDE used for enterprise application development:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 58

Use slides 57 and 58 to explain the utility of IDEs in enterprise application development.

IDEs simplify the process of application development. They provide a skeletal structure to the application. IDEs are aware of code syntax, hence, prompt the developer in case of trivial code syntax errors. They have various tools for coding and debugging the application.

IDEs provide a directory structure useful for application deployment, with essential deployment descriptors. They provide tools for database connections and network communication which makes the task of the developer simpler.

Slide 59

Let us summarize the session.

Summary

- ❑ Enterprise JavaBeans are application components which implement business logic of the application.
- ❑ There are three types of enterprise beans – Session beans, Entity Beans, and Message-driven beans.
- ❑ Session beans can be stateless, stateful, and singleton, they are synchronously invoked by the application client.
- ❑ Message-driven beans are enterprise beans which are asynchronously invoked through JMS messages.
- ❑ Entity beans are used to implement persistence in enterprise applications.
- ❑ Entities are enterprise beans which implement database operations, they are deprecated and entity persistence to database is implemented by Java Persistence API in EJB 3.0.
- ❑ Enterprise beans are deployed in a container which provides services such as transaction management, component security, naming services, and so on.
- ❑ Enterprise beans can be accessed through business interface view or no interface view of the application.
- ❑ Annotations are metadata used in the application for deployment and dependency injection.
- ❑ Enterprise beans are packaged into EAR file.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 2 59

Using slide 59, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also familiarize yourself with the concepts of Session Beans which will be explained in the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 3 – Session Beans

3.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

3.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe Session beans
- Describe the characteristics of Session bean
- List the different types of Session beans
- Explain Stateless Session beans
- Describe the lifecycle of Stateless Session beans
- Explain the process for developing a Stateless Session bean in NetBeans IDE
- Describe the types of communication with Session beans
- Explain asynchronous communication in Stateless Session bean

3.1.2 Teaching Skills

To teach this session successfully, you should have in-depth knowledge of different types of session beans and their characteristics. You should aware yourself with the lifecycle of Stateful Session bean and the process of deploying in the application container.

Further, you should aware on how the application clients communicate with the Stateful Session beans.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- Describe Session beans
- Describe the characteristics of Session bean
- List the different types of Session beans
- Explain Stateless Session beans
- Describe the lifecycle of Stateless Session beans
- Explain the process for developing a Stateless Session bean in NetBeans IDE
- Describe the types of communication with Session beans
- Explain asynchronous communication in Stateless Session bean

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 2

Tell them that they will be introduced to Session bean components. They will learn about characteristics of different types of Session beans. Tell them that though there are three different types of Session beans, the focus of the session will be Stateless Session beans.

Tell them that in this session, the lifecycle of Stateless Session beans will be discussed and how developers can create Stateless Session beans through NetBeans IDE. Finally, the session discusses the mechanisms used by application clients to communicate with the Stateless Session beans.

3.2 In-Class Explanations

Slide 3

Let us understand the use of Session beans in enterprise applications.

Introduction

- ❑ Session beans represent business processes that are used to handle business logics for the application.
- ❑ A business logic can be:
 - Performing addition on two numbers
 - Connecting to a database
 - Performing transaction on a bank account
 - Invoking other Session beans or Message-driven beans

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 3

Use slide 3 to explain the use of Session beans in enterprise applications.

The Java EE platform supports different types of EJB components based on the functionality and integration with other components in the application. Enterprise JavaBeans are categorized into various types of bean development. They are Session beans and Message-driven beans.

Session beans represent business processes that are used to handle business logics for the application. The business logic can be performing addition on two numbers, connecting to a database, performing transaction on a bank account, or invoking other Session beans or Message-driven beans.

The business logic of the application is specific to the context of the application. Developers create appropriate Session beans and other EJB components to implement the business logic. Session beans in turn can invoke other EJB components, database components, and so on.

Tips:

Session beans frequently manage the interactions within persistent data through Entity beans to achieve the specific goal.

Slides 4 and 5

Let us understand the characteristics of Session beans.

Session Beans 1-3



- Are reusable Java components.
- Execute on the server-side in an EJB container.
- Are deployed on Java-enabled application servers.
- Provide services to their clients which can access them programmatically.
- Are conversational.

A conversation is an interaction between a bean and a client and it is composed of a number of method calls between them.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 4

Session Bean 2-3



- Session bean components implement `javax.ejb.SessionBean` interface.
- Session beans are not persistent across system failures.
- Lifetime of Session bean:
 - An instance of a Session bean is alive as long as the client is present.
 - Hence, the lifetime of a Session bean is equivalent to the lifetime of a client.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 5

Use slides 4 and 5 to explain the characteristics of Session beans.

Session beans are reusable application components that are deployed in the EJB container on the application server. The application server can be any of the Java enabled servers which communicate with application clients and provide services to the client.

The clients can access the Session beans programmatically.

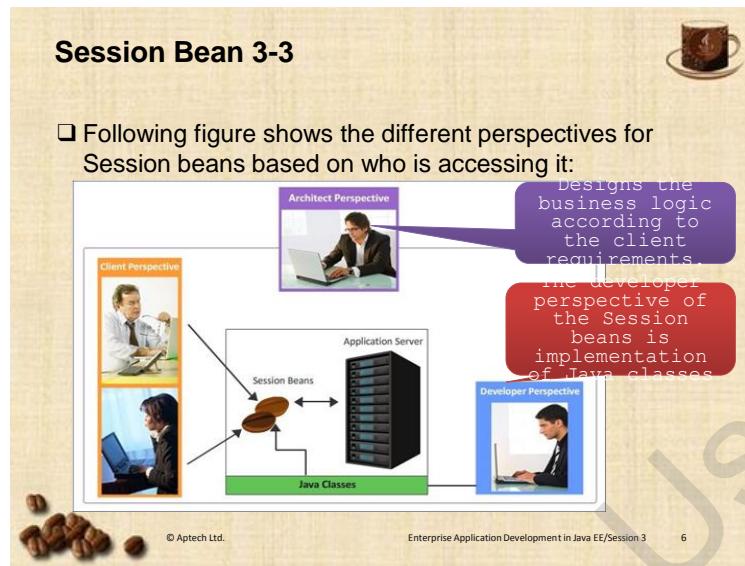
Session beans are conversational which means the information exchanged between the clients request and EJB is stored within the context of the request.

Tell them that as Cookies can be used to maintain the client data in Web applications. Similarly, Session beans can hold the state of the client during the method calls.

Session beans can be accessed by both local clients and remote clients. The access to the Session beans is through the business interface provided by the developer for the bean. The lifecycle of a Session bean begins when there is a client request from the bean and ends when the client request is serviced. All Session beans implement the interface `javax.ejb.SessionBean`.

Slide 6

Let us understand different perspectives of a Session bean.



Use slide 6 to explain different perspectives of a Session bean.

There are three different people who are associated with a Session bean in the enterprise application.

The application architect outlines the business logic to be implemented for the application. Based on the application context, the architect defines the Session beans to be created for the application. Therefore, the application architect's perspective of Session bean is in relation to all other components of the application.

Based on the definition provided by the application architect, the application developer writes code for the EJBs and implements their functionality. Developer's perspective is limited to the implementation of a Session bean and to ensure that the Session bean functions according to the specification provided by the application architect.

The client perspective of the Session bean is the service provided by it. The interface exposed to the application clients is from their perspective and the clients use Session bean services accordingly.

Slide 7

Let us understand the types of Session beans.

Types of Session Beans 1-4

- ❑ Based on the lifetime of the Session bean, there are three variants of Session beans:
 - Stateless Session beans
 - Stateful Session beans
 - Singleton Session beans

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 7

Use slide 7 to explain different types of Session beans.

The main difference between the Session bean and JavaBeans is the lifetime of the Session bean. An instance of a Session bean performs a task on behalf of the client. In other words, it represents client on the server-side and acts as an extension of client on the server. An instance of a Session bean is alive as long as the client is present. Hence, the lifetime of a Session bean is equivalent to the lifetime of a client. It maintains the conversation state for the client.

Conversation can be defined as the period between the client accessing a Session bean instance and client invoking the remote method on the instance.

There are three types of Session beans according to EJB 3.0 specification – Stateless Session beans, Stateful Session beans and Singleton Session bean. All three types of beans can communicate with the application clients.

Stateless session beans do not maintain the conversational state of communication with the client. Whenever a method of the Stateless Session bean is invoked, it uses the instance variables of the bean, but the values of these variables are retained only for the duration of method invocation. After the method is completed, the values are not retained, that is, they are deleted by the container.

Then, discuss some of the situations where the use of Stateless Session bean can be done. They are as follows:

- SalaryCalculator
- CardVerification
- MoneyConverter

Stateful Session bean maintains the conversational state of communication with the application client. This means that they store information obtained from the specific client between the method invocations. In other words, Stateful Session beans are responsible to retain the client state across various method conversations.

State of the instances of a Session bean cannot be saved in a database or a file system. Hence, Session beans are not persistent, but they can perform database operations.

Then, discuss some situations which identify the need of Stateful Session bean that are as follows:

- Shopping Cart
- Online Banking Transactions
- Order Processing System

Singleton Session bean is instantiated only once in the entire application. They are accessed concurrently by multiple clients and they are similar to Stateless Session beans.

Slide 8

Let us understand where a Stateless Session bean is used.

Types of Session Beans 2-4

❑ Stateless Session beans

- Does not maintain the client state on the server during conversation which means it does not store the value of the instance variables associated with the client conversation.
- After the method execution is completed, the values are not retained, that is, they are deleted by the container.
- Some of the situations which do not require the client's data to be maintained and are suitable for Stateless Session bean development are:
 - SalaryCalculator
 - MoneyConverter
 - CardVerification

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 8

Use slide 8 to explain the utility of Stateless Session beans in enterprise applications.

Developers use Stateless session beans in instances where the application need not maintain the state of communication with the client.

Stateless session beans do not maintain the conversational state of communication with the client. Whenever a method of the Stateless Session bean is invoked, it uses the instance variables of the bean, but the values of these variables are retained only for the duration of method invocation. After the method is completed, the values are not retained, that is, they are deleted by the container.

Then, discuss some of the situations where the use of Stateless Session bean can be done. The examples given on slide 8 are on similar lines where the application would receive input from the user interface and provide processed output.

They are as follows:

- SalaryCalculator
- CardVerification
- MoneyConverter

Salary calculator can take variables such as Basic pay, Daily allowance, House Rent Allowance and other parameters as input and the Session bean may implement the logic of calculating the salary based on these parameters and return the total Salary.

Money converter is an application where the application performs currency conversions such as Dollar to Pound, and vice versa. The input might to the currency converter component must be a numeric value which will be converted into its equivalent value based on the selected currency.

Similarly, the card verification is a process where the card number is provided as input and the validity of the card may be checked. The check needs to be done against the rules such as how many digits should be there in a valid card number, the valid digits with which a credit card number may begin, and so on.

Slide 9

Let us understand what Stateful Session beans are.

Types of Session Beans 3-4

□ Stateful Session beans

- Maintain the state of the clients on the server which means they store information obtained from the specific client between the method invocations.
- Some of the situations where it is necessary to store the client's information to maintain its identity and are suitable for Stateful Session bean are:
 - Shopping cart
 - Online banking
 - Product order processing system

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 9

Use slide 9 to explain the basic functionality of Stateful Session beans. Developers use Stateful session beans when the application has to store the conversational state of the Session beans.

Stateful Session bean maintains the conversational state of communication with the application client. This means that they store information obtained from the specific client between the method invocations. In other words, Stateful Session beans are responsible to retain the client state across various method conversations.

State of the instances of the Session beans cannot be saved in a database or a file system. Hence, Session beans are not persistent, but they can perform database operations.

Then, discuss some situations which identify the need of Stateful Session bean that are as follows:

- Shopping Cart
- Online Banking Transactions
- Order Processing System

Tips:

Stateful Session beans are used when the application has a user login through which the client has to access the application. When the user logs in with his/her username and password the application returns user specific information and when the user logs out the user specific information is persisted onto the permanent storage.

Shopping cart, online banking application, and product order processing are three such applications where the user information is stored by the application. Each user accessing the application is associated with an instance of Stateful Session bean, which is activated when the specific user accesses the application.

Slide 10

Let us understand Singleton Session beans.

Types of Session Beans 4-4

❑ Singleton Session bean

- Are similar to Stateless Session bean.
- Are instantiated once for the entire application.
- A single instance of the Singleton bean is shared by all the clients.
- Some of the scenarios where Singleton Session bean can be used for:
 - Executing initialization
 - Clean tasks of the application when it shuts down.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 10

Use slide 10 to explain Singleton Session beans.

Singleton Session beans are instantiated only once during the lifecycle of the application. This instance may or may not be retained throughout the lifecycle of the application.

Singleton Session beans also do not retain the state of the Session bean, it can be uniformly accessed by all the application clients. Developers use Singleton Session beans to implement tasks such as executing all the initialization tasks, logging all the application data, and performing application clean up tasks when the application has to shut down.

Slide 11

Let us understand the characteristics of Session beans.

Characteristics of Session Beans 1-2




© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 11

- ❑ Regardless of the type of Session Beans developed in the enterprise application, they possess the following characteristics:
 - Session beans are short-lived objects.
 - Each Session bean instance is associated with a single client.
 - Session bean instance cannot be shared by multiple clients.
 - Session beans represent the state of communication between the client and bean.
 - Session beans are instantiated and managed by the EJB container.
 - Session beans support synchronous and asynchronous communication.
 - Session beans are non-persistent.
 - Session beans can implement transaction boundaries and security mechanisms.

Use slide 11 to explain the characteristics of Session beans.

Tell them that regardless of the type of Session Beans developed in the enterprise application, they possess the following characteristics:

- Session beans are short-lived objects. They are removed when the client removes the bean instance or when the EJB container shuts down or crashes.
- They are instantiated only when a client requests for an application service. They execute on behalf a client and cannot be shared by more than one client using multiple threads.
- Each Session bean is accessed by a single client; a Session bean cannot service multiple requests simultaneously.
- Session beans are managed by the container. This means that the execution of the session bean is controlled by the container.
- Session beans can communicate with the client both synchronously and asynchronously.
- Session beans can implement transactions and also implement security mechanisms in the application.

Tips:

Application containers generally maintain a pool of Stateless Session beans which provide a particular service. When a client request arrives, the container assigns it to a particular bean instance and deallocates the bean. Finally, the container returns it to the pool after the request is serviced.

When an application client and bean communicates in a synchronous mode, the client sends a request to the server and waits for response without performing any other task. In case of asynchronous communication, the client sends a request and does not wait until it receives a response. The client in the meanwhile performs other tasks which do not have any dependency on the response expected from the server.

Slide 12

Let us understand the scenarios in which a developer can use Session beans.

Characteristics of Session Beans 2-2

When should the Session beans be used in enterprise applications?

- When the application has to retain the state across multiple method invocations.
- When there is a requirement of communication between the client and other components of the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 12

Use slide 12 to explain in what scenarios developers can choose to use Session beans.

The Session bean is used in the application when any of the following conditions hold good:

- When the application has to retain the state across multiple method invocations.
- When there is a requirement of communication between the client and other components of the application.

Then, you can discuss a scenario of the shopping cart that is employed by many Web-based, applications is a typical use for a Session bean. It is created by the online shopping application only when an item is selected by the user. When selection is completed, the item prices in the cart are calculated, the order is placed, and the shopping cart object is released, or freed. A user can continue browsing merchandise in the online catalog, and if the user decides to place another order, a new shopping cart is created.

Often, a Session bean has no dependencies on or connections to other application objects. For example, a shopping cart bean might have a data list member for storing item information, a data member for storing the total cost of items currently in the cart, and methods for adding, subtracting, reporting, and totaling items.

On the other hand, the shopping cart might not have a live connection to the database at all.

Slides 13 to 15

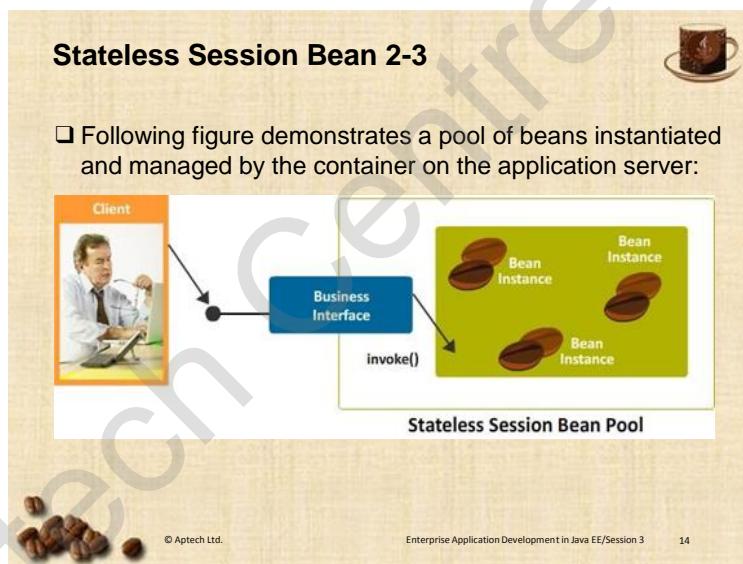
Let us understand how Stateless Session beans are managed by the application container.

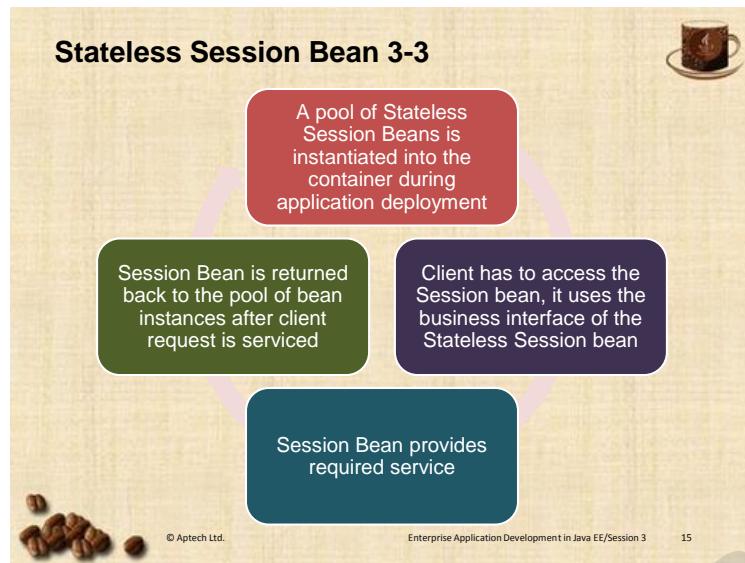
Stateless Session Bean 1-3



- ❑ Is invoked by the client when it requires service of an application server.
- ❑ Does not maintain the conversational state of the session with the client.
- ❑ Are maintained in a pool by the container.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 13





Use slides 13 to 15 to explain the process of managing the Stateless Session beans in the application container.

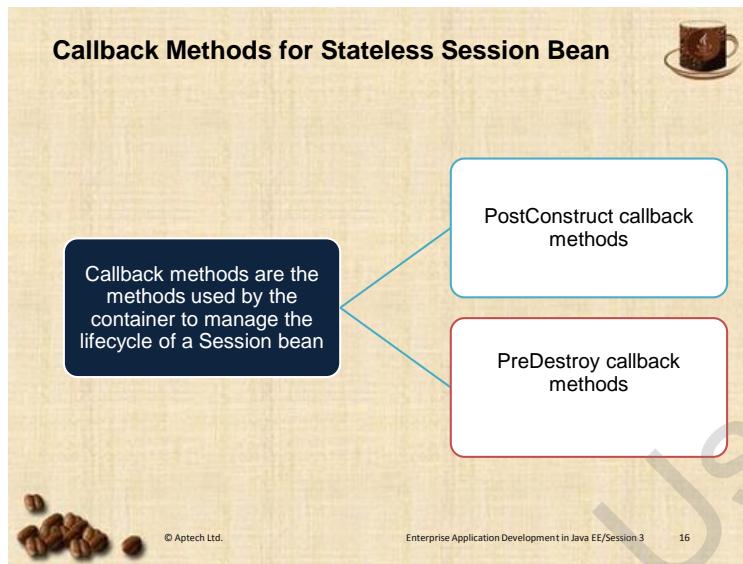
Use slide 13 to explain the invocation of Stateless Session bean by the client.

Then, using slides 14 and 15 to explain the figure that demonstrates the pool of beans instantiated and managed by the container. Tell them that a pool of Stateless Session beans is instantiated into the container during application deployment. When a client has to access the Session bean, it uses the business interface of the Stateless Session bean. Through the business interface defined, the application client accesses the Session bean from the pool of available beans. The Session bean then provides the required service. The bean is returned to the pool of available bean instances after the client request is serviced.

The client requests do not actually reach the Session bean, but they reach the container in which the Session bean is deployed. The container assigns the client request to an appropriate instance of bean. The container manages which Session bean has to be invoked. If all the instances are assigned to other clients, the container may instantiate a new bean for the new client request. These choices are made by the container based on its configuration.

Slide 16

Let us understand the callback methods associated with Session beans.



Use slide 16 to explain the callback methods associated with Stateless Session beans.

Callback methods along with the Session beans are used to implement various operations during the lifecycle events of the Session bean.

There are two types of callback methods associated with Stateless Session beans – **PostConstruct** methods and **PreDestroy** methods. Lifecycle callback methods are invoked by the application container.

PostConstruct methods are invoked after the Stateless Session bean has been instantiated into the application container.

PreDestroy methods are invoked before the EJB is removed from the container.

As EJB 3.0 supports annotations, the lifecycle methods are prefixed with `@PostConstruct` and `@PreDestroy` annotations respectively.

Tips:

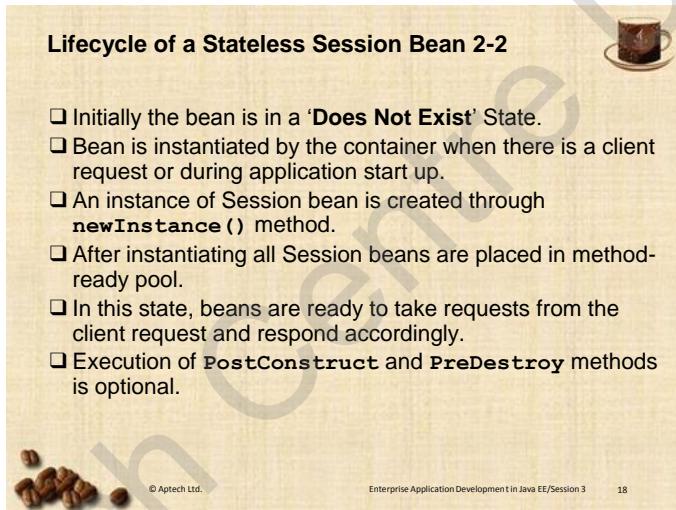
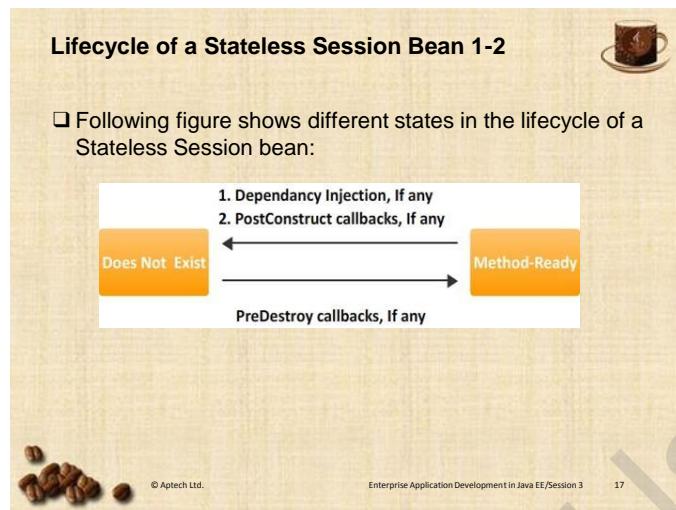
The callback methods are annotated with `@PostConstruct` and `@PreDestroy` annotations.

- The return type of the callback methods should be **void**.
- The callback methods can be prefixed access modifiers - **public, private, protected, and package private**.
- Callback methods can be final methods.

Lifecycle callback methods are not defined in the business interface of the EJB.

Slides 17 and 18

Let us understand the lifecycle of Stateless Session beans.



Use slides 17 and 18 to explain the lifecycle of Stateless Session beans. There are two states in the lifecycle of the Stateless Session bean.

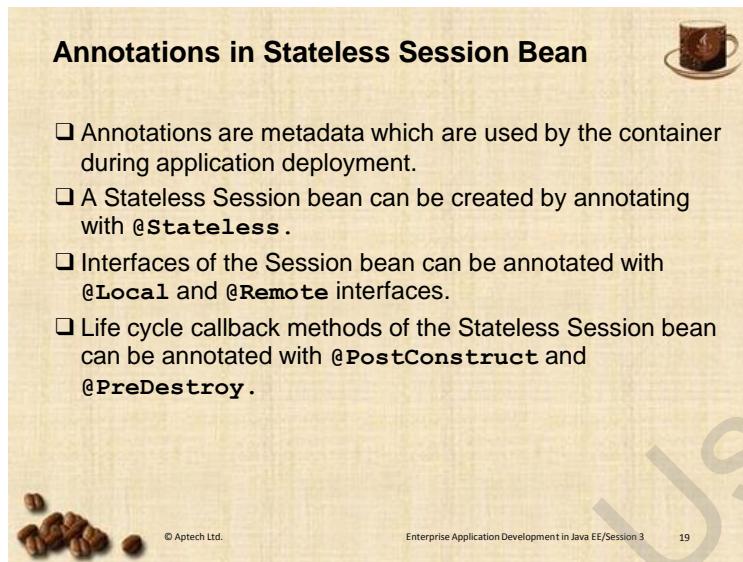
Initially the Stateless Session bean is in a '**Does Not Exist**' state. The bean remains in this state until it is instantiated by the application container. This process of instantiating a bean into the application container is also termed as loading the Session bean into the application container.

After loading the Session bean, the container invokes the `PostConstruct` callback methods if any and also performs dependency injection.

The Session bean comes in the '**Method Ready**' state after it is instantiated, in this state, the bean can accept client requests and service them. The Session bean remains in this state, till the bean is removed from the container. When the container intends to remove the Session bean from the container it executes the `PreDestroy` callback methods. After executing the `PreDestroy` methods, the Session bean transits to '**Does Not Exist**' state.

Slide 19

Let us understand the annotations associated with Stateless Session beans.



The slide has a parchment-like background with a small coffee cup icon in the top right corner and coffee beans at the bottom left. The title 'Annotations in Stateless Session Bean' is at the top left. Below it is a bulleted list of annotations:

- ❑ Annotations are metadata which are used by the container during application deployment.
- ❑ A Stateless Session bean can be created by annotating with **@Stateless**.
- ❑ Interfaces of the Session bean can be annotated with **@Local** and **@Remote** interfaces.
- ❑ Life cycle callback methods of the Stateless Session bean can be annotated with **@PostConstruct** and **@PreDestroy**.

At the bottom center, it says 'Enterprise Application Development in Java EE/Session 3' and '19'.

Use slide 19 to explain the annotations associated with Stateless Session beans. Annotations are metadata associated with application components. Annotations are used by the container during application deployment.

Stateless Session beans have various annotations associated with them.

@Stateless annotation – It is used to indicate that the bean class which is annotated with **@stateless** annotation implements a Stateless Session bean.

Each Session bean is associated with an interface. The interface can be local or remote. The local interface is annotated with **@Local** and the remote interface is annotated with **@Remote**.

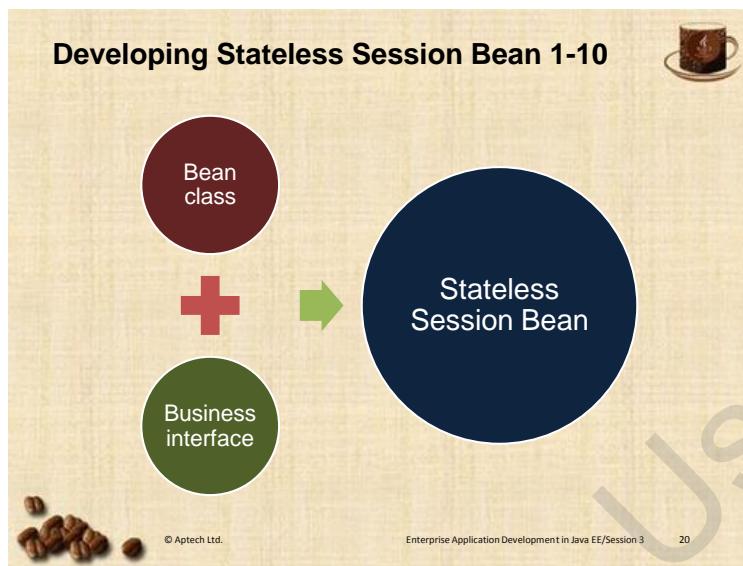
There are two types of callback methods associated with Stateless Session beans – PreDestroy and PostConstruct methods. These methods can be defined within the bean class, but it is essential to annotate them with **@PreDestroy** and **@PostConstruct** annotations.

Tips:

Session beans are allocated to the clients from the pool based on the client invocation. Stateless Session beans are extremely performance-oriented beans and small amount of beans created and stored in the pool, helps the application to serve concurrent clients.

Slide 20

Let us understand the development process of Stateless Session bean.



Use slide 20 to explain the development process of Stateless Session bean.

Explain that the development process of Stateless Session bean contains two components:

- Bean class
- Business interface

A Bean class refers to the Java class implementation of the business logic expected from the bean. It has a set of instance variables and methods defined to accomplish the task expected.

Business interface refers to the set of methods which are exposed to the client. The business interface is accessed by the client applications to invoke the methods of the Stateless Session bean.

Apart from the business interface, the Session bean can be accessed through local interface or No-interface view.

Slide 21

Let us understand the rules followed for defining a Stateless Session bean.

Developing Stateless Session Bean 2-10

Following are some of the rules to be followed while developing a Stateless Session bean:

- There should be at least one business interface for the Session bean.
- Session bean class cannot be declared as **final** or **abstract**.
- Session bean class should implement a no argument constructor.
- Session bean class can be a subclass of another Session bean or a POJO class.
- Business and life cycle callback methods are defined in the Session bean class or **super** class.
- Business method names present in the business interface and in the Session bean class must not begin with **ejb**.
- Business methods should be declared as public and cannot be declared as **final** or **static**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 21

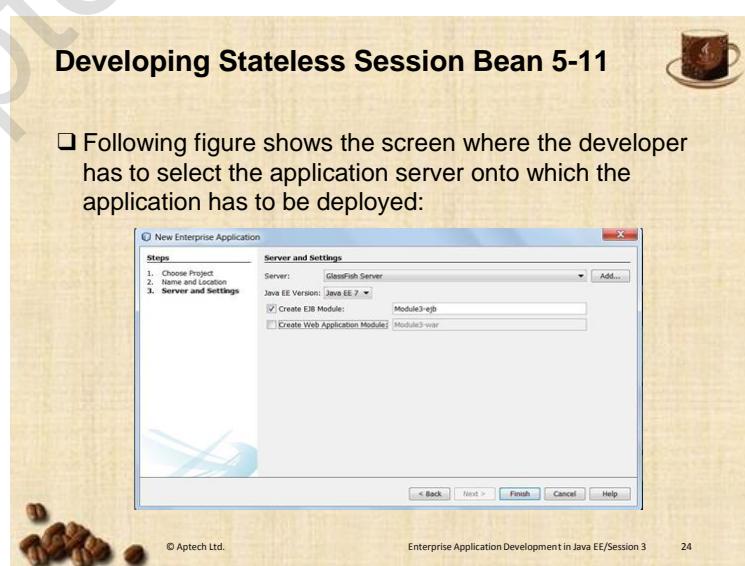
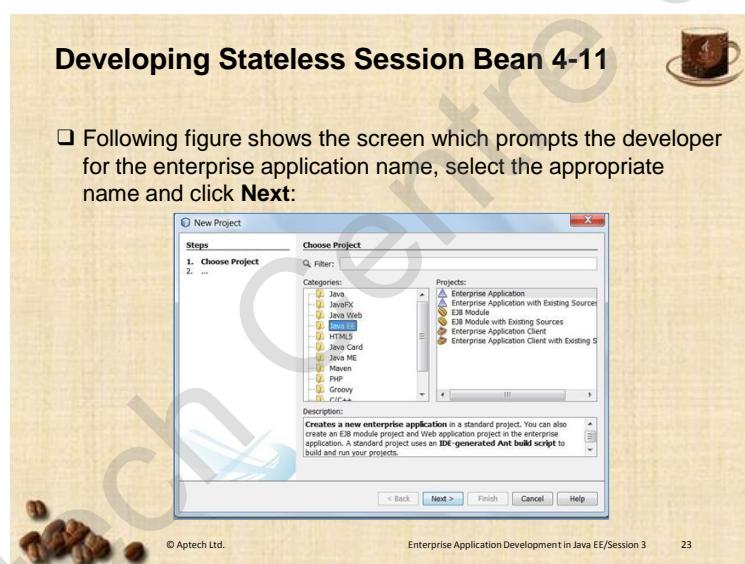
Use slide 21 to explain the rules to be followed for defining a Stateless Session bean.

Explain the rules to be followed while developing the Stateless Session bean for an enterprise application.

- A Session bean can be accessed only through an interface; therefore it is essential to have at least one business interface for each Session bean.
- A Session bean class should implement all the business methods, therefore Session bean class cannot be declared **abstract** or **final**.
- Every Session bean class should have a default constructor, apart from the default constructor Session bean class can have additional constructors also.
- A Session bean class can be part of a hierarchy of classes and can also be a sub class of another class.
- In case Session class is part of a class hierarchy, the callback methods can be defined either in the bean class or in the superclass of the bean class.
- The keyword '**ejb**' cannot be used as a prefix to the business methods in the bean class. It cannot be used as a prefix to EJB class name.
- The business interface has to be accessed by the application clients therefore, it cannot be declared as **private** or **protected**.

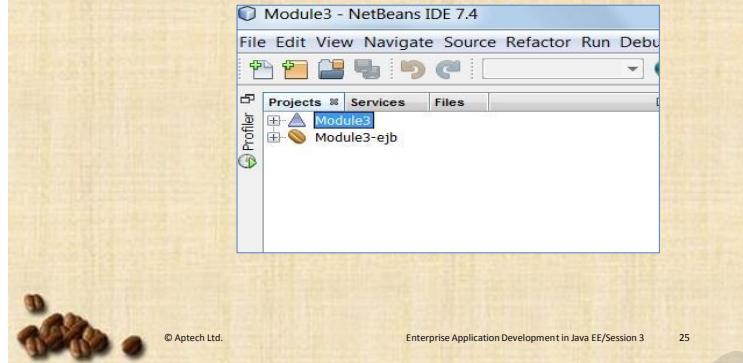
Slides 22 to 27

Let us understand the process of creating Stateless Session beans through NetBeans IDE.



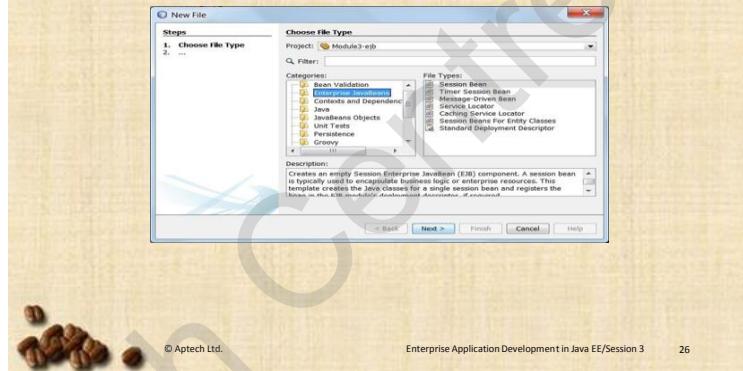
Developing Stateless Session Bean 6-11

- Following figure shows the Project in the **Project** window after it is created by the IDE:



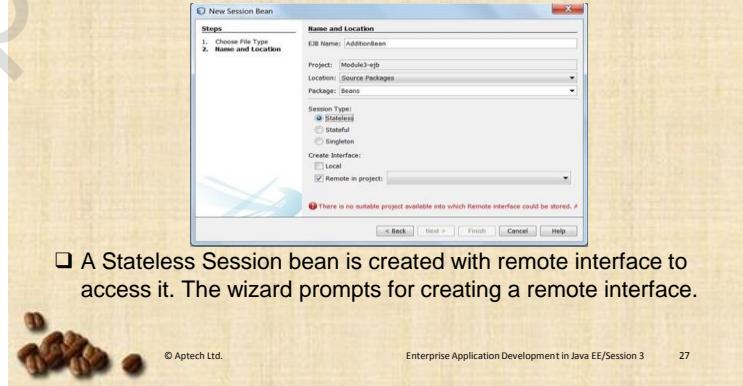
Developing Stateless Session Bean 7-11

- Following figure shows how to create a Session bean after the Project has been created in the IDE:



Developing Stateless Session Bean 8-11

- Following figure shows the wizard to select options for the Stateless Session bean:



- A Stateless Session bean is created with remote interface to access it. The wizard prompts for creating a remote interface.

Use slides 22 to 27 to demonstrate the process of creating Stateless Session beans using NetBeans IDE.

Tell them NetBeans IDE is a Java compatible integrated development environment used by developers to create applications. The Netbeans IDE provides complete support for the latest Java EE 7 APIs and standards. It provides wizards to create the enterprise application components quickly with default template.

All the components of the enterprise application developed using NetBeans IDE have to be created as a part of a '**Project**'.

Then, explain the steps to create the Stateless Session bean. Firstly, create a new Project by choosing appropriate options from the File menu as shown on slide 22.

Use slide 23 to demonstrate the options to be chosen while creating the enterprise application. Choose **Java EE** from the categories and the type of Project should be '**Enterprise Application**'.

Use slide 24 to demonstrate how the developer has to select the application server onto which the application has to be deployed. If there are multiple versions of Java EE, then the developer has to choose the version the developer intends to use for the application. Choose **Java EE → Enterprise Application** as shown on slide 24.

Also, choose what modules of the application are to be created. The IDE by default creates an EJB module and Web module.

Use slide 25 to demonstrate the creation of the enterprise application named '**Module3**' and its corresponding EJB module and Web module created in the default directory structure.

Now, create a Stateless Session bean in the EJB module for the application.

Use slide 26 to demonstrate that the developers can add a Session bean to the application. To do so, select the EJB module and select **New File** from the **File** menu. The **Choose File Type** dialog box is displayed as shown on slide 26.

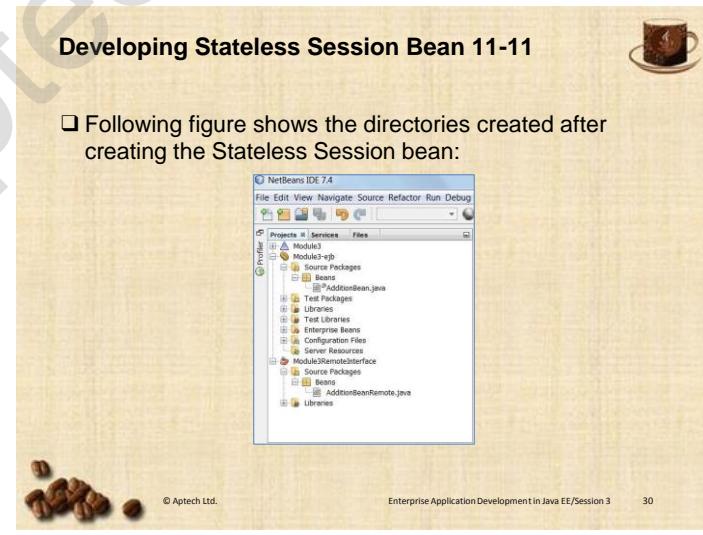
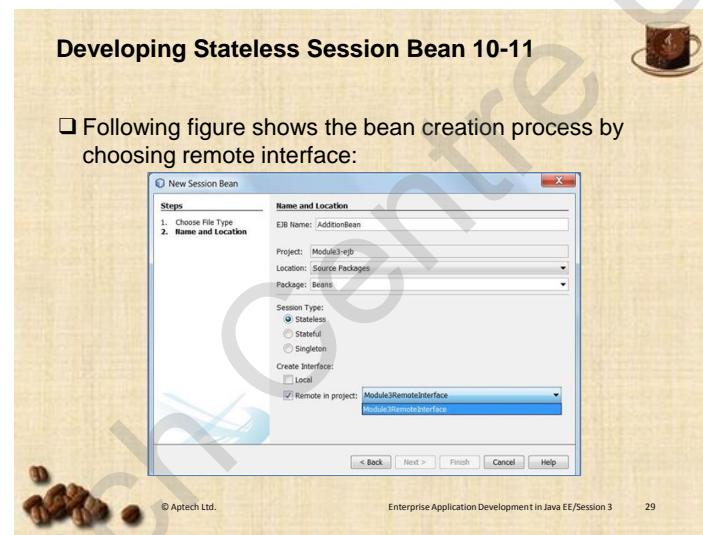
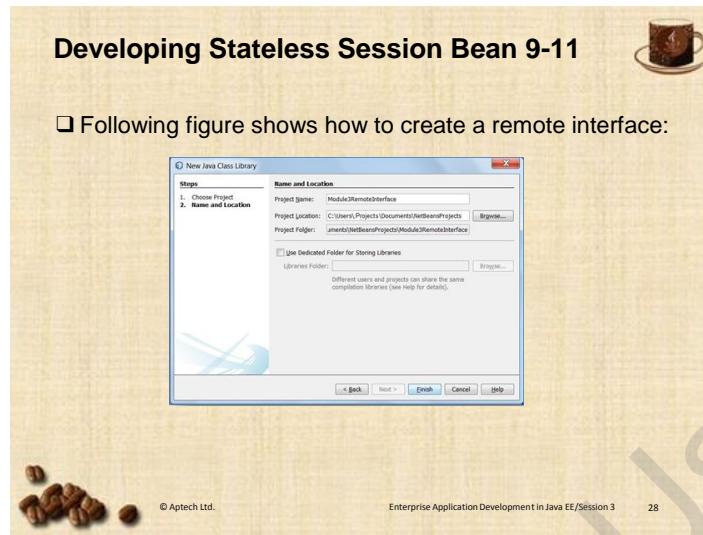
Use slide 27 to explain the process of creating a Session bean and creating a corresponding interface. Choose **Enterprise JavaBeans → Session Bean** in the current screen and click **Next**.

Then, select the **Session Type** to select the type of bean to be created. Select options **Stateless** for the Stateless Session bean. First, provide the name of the bean and a package in which the bean need to be packaged. It is a good practice to keep the beans in different directories according to their functionality.

Then, select **Local or Remote in Project** for providing the interface to the bean. When remote interface is selected, the application prompts for a suitable project to store the remote interface. Hence, firstly create a remote interface.

Slides 28 to 30

Let us understand the process of creating a remote interface.



Use slides 28 to 30 to explain the steps involved in creating a remote interface.

To create a remote interface, create a new Java Class Library project in NetBeans IDE by clicking **File → New Project → Java → Java Class Library** from the **New Project** dialog box.

Then, provide an appropriate name to the remote interface; choose a location for the interface as shown on slide 28. The remote interface is created.

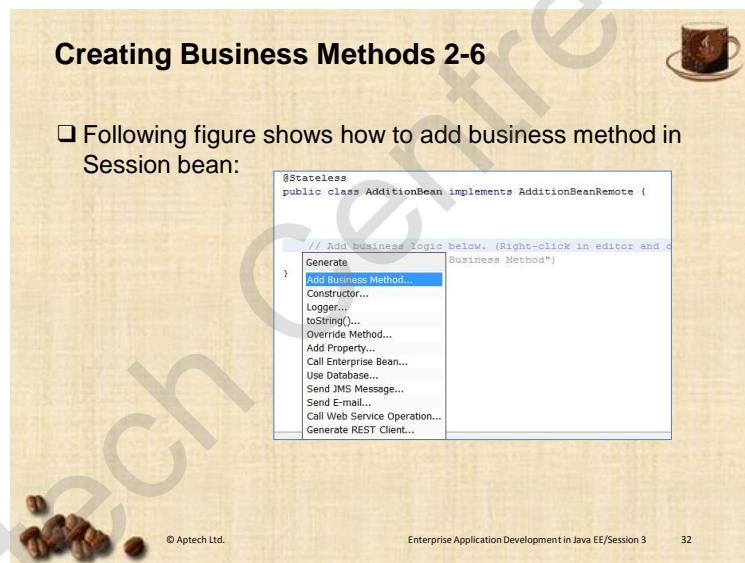
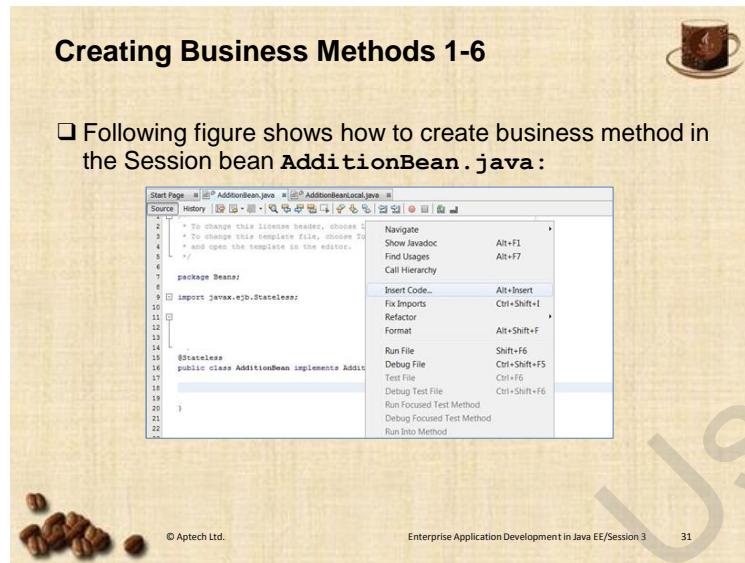
Use slide 29 to demonstrate the process of choosing the remote interface for the Session bean. As shown on slide 29, select the appropriate remote interface class from the Remote in project dropdown list and complete the process of creating the Stateless Session bean.

Then, use slide 30 to explain the directories created in the enterprise application. The IDE defines various directories in the directory structure of the created application. The bean classes are created in the '**Source Packages**' folder of the application. Apart from the '**Source Packages**' folder the application has **Test Packages**, **Libraries**, **Configuration Files**, and **Server Resources**.

So, far the skeletal structure of the enterprise application has been created. Now, the developer has to write code for the implementation of Session bean class.

Slides 31 to 36

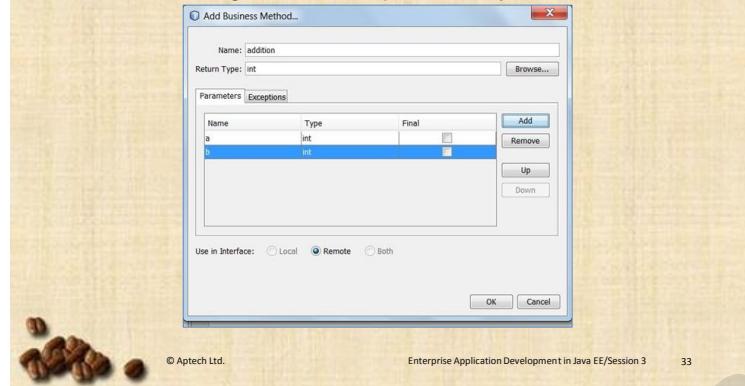
Let us understand and define the business logic of the Session bean by writing the code.



Creating Business Methods 3-6



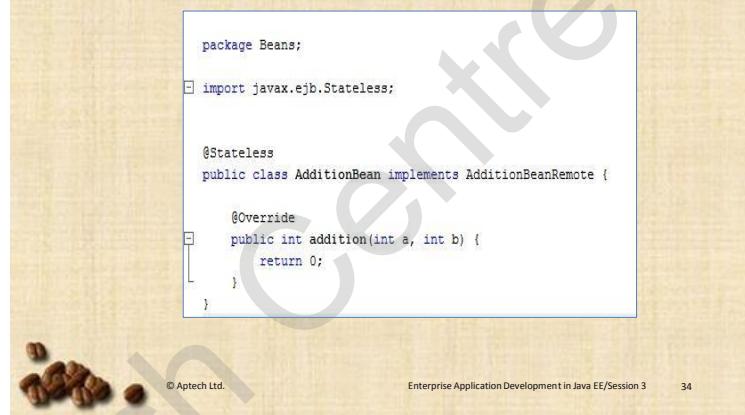
- Following figure shows how to configure the business method through the wizard provided by NetBeans IDE:



Creating Business Methods 4-6



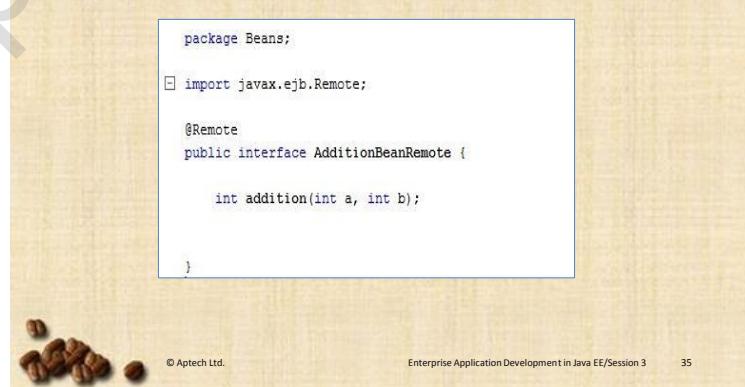
- Following figure shows the code created by the wizard:

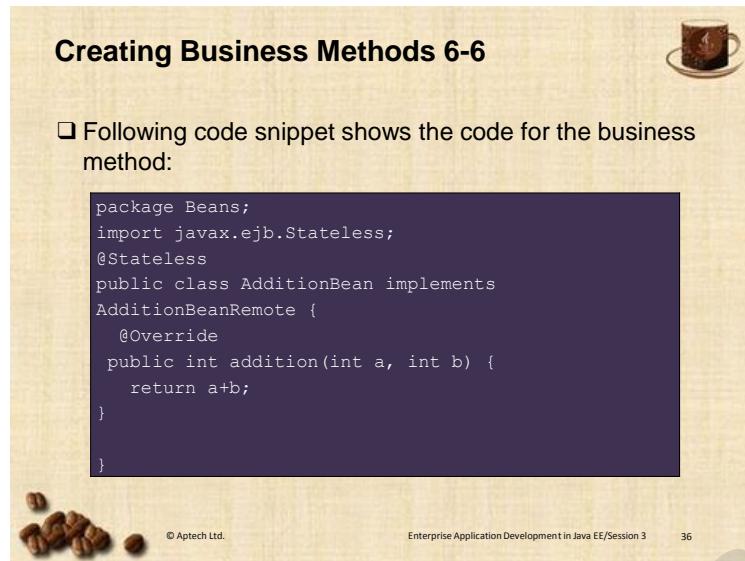


Creating Business Methods 5-6



- Following figure shows the modifications to be made to the remote interface of the Session bean:





Creating Business Methods 6-6

Following code snippet shows the code for the business method:

```
package Beans;
import javax.ejb.Stateless;
@Stateless
public class AdditionBean implements
AdditionBeanRemote {
    @Override
    public int addition(int a, int b) {
        return a+b;
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 36

Use slides 31 to 36 to explain the process of adding code to the Stateless Session bean class.

When a Stateless Session bean class is created through the IDE, the IDE creates certain skeletal code with the class name and the basic libraries.

Double-click the bean file, **AdditionBean.java** to open in the code editor and then, add bean implementation code to it. In the editor, right-click in the area where the business methods are defined and it will pop out a menu as shown in the figure given on slide 31.

In the code, choose '**InsertCode**' option. It will lead to another menu as shown on slide 32. Then, choose '**Add Business Method**' for the current implementation of Session bean. Tell them that IDE provides other options as well. Choosing each option will add relevant code in the code editor. The developer can add additional code according to the application requirement.

Use slide 33 to demonstrate the Add Business Method dialog box that can be used to define the business methods. Tell students that the developers can define the prototype of the business method, its return types, its parameters, name of the method, and so on using the dialog box.

Use slide 34 to show the code written for the business method. In this demonstration, a Session bean has been created which implements the function of a **CalculatorBean**. A business method is defined that accepts two **int** parameters and returns the added value of the two parameters. The code shown on slide 34 is the default code for the business method provided by IDE.

The developer needs to add the business code to the generated method definition.

Use slide 35 to explain how the remote interface of the Session bean has changed after the business method is added to the Session bean. The developer has not explicitly modified the remote interface; these changes are implicitly done by the IDE.

Use slide 36 to explain the modifications made in the Session bean code. Shows them the package to which the current bean is located.

The annotation `@Stateless` implies that it is a stateless bean so that it can be appropriately deployed. The implementation of a Stateless bean is imported from `javax.ejb.Stateless` package. The bean implements the remote interface `AdditionBeanRemote`.

In-Class Question:

After you finish explaining the development process of Stateless Session bean, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



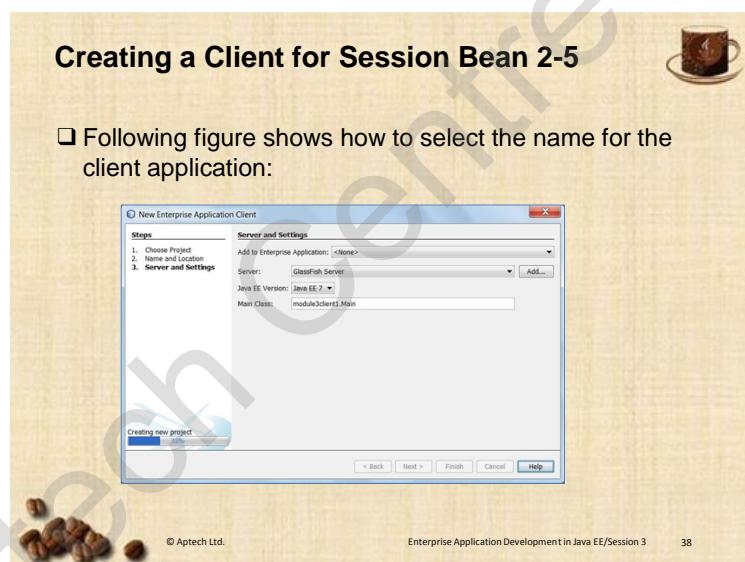
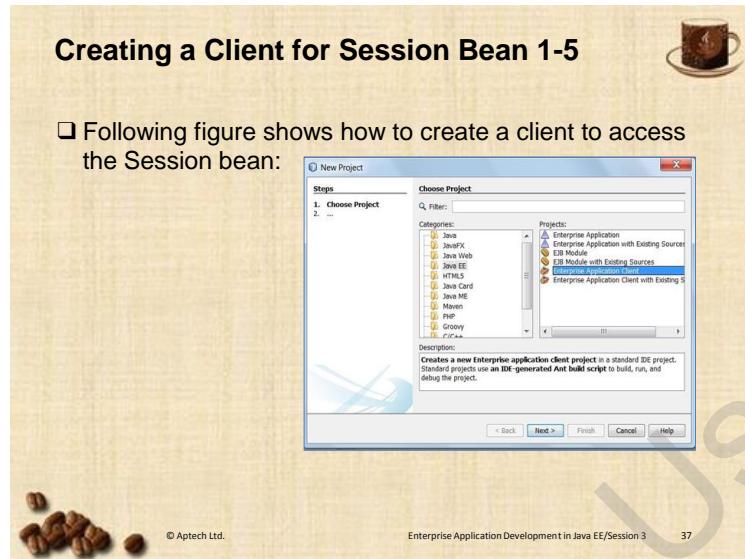
Consider a situation, where the code shown on slide 36 to be changed, if the bean has to perform an addition of three integer values. How can you achieve this in the CalculatorBean class?

Answer:

The developer has to define an overload method which accepts three integer parameters from the user.

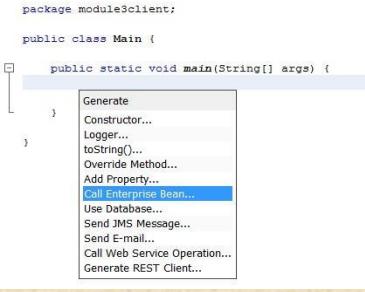
Slides 37 to 41

Let us understand how a client application has to be created to access the Session bean.



Creating a Client for Session Bean 3-5

Following figure shows how to invoke a Session bean from the client application:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 39

Creating a Client for Session Bean 4-5

Following figure shows the wizard which allows choosing the Session bean to be accessed by the client:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 40

Creating a Client for Session Bean 5-5

Following code snippet demonstrates the client code through which the session bean is accessed:

```
package module3client;
import Beans.AdditionBeanRemote;
import javax.ejb.EJB;
public class Main {
    @EJB
    private static AdditionBeanRemote additionBean;
    public static void main(String[] args) {
        System.out.println("Result:"+
additionBean.addition(4,2));
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 41

Following figure shows the output after deploying and running the application client:



Use slides 37 to 41 to demonstrate how a client application can access the Stateless Session bean.

Tell them that they need to create an application client which can access the Stateless Session bean. The client can be a plain Java class or Java Servlet.

To create an application client, create a new client application by selecting **Enterprise Application Client** under **Projects** with options as shown on slide 37.

Then, provide a name to the client application. Choose **Module3client1** as the name for the application client **Main** class and other options as shown on slide 38.

Use slide 39 to demonstrate how to add the access code to the application client. Developer has to right-click in the client code and choose the option '**Call Enterprise Bean**' to add Session bean code. This action will present a list of available interfaces; choose the interface of the Session bean you want to access.

Using slide 40, select `AdditionBean` class and modify the application client code. Choosing the relevant interface will add code to the application client. As shown in the code, the remote interface is imported into the client module through `import Beans.AdditionBean` accessing the `AdditionBean` class used to invoke the method.

Use slide 41 to explain how to add code to the application client `main()` method. After choosing the remote interface a reference to the remote interface is added to the application client code. Using this reference, the methods of the remote interface are accessed. Then, show the output after the execution of the code as shown on slide 41. The application client passes two parameters to the `add()` method. The Session bean returns the sum of the two values to the client.

Slide 42

Let us understand the configuration and deployment of the Stateless Session bean.

Configuring and Deploying Session Beans 1-2

- A Session bean can be configured through deployment descriptor and annotations.
- Deployment descriptor is a declarative XML file.
- ejb-jar.xml** is the deployment descriptor for the enterprise application.
- Deployment descriptor determines how the EJB container manages the bean when deployed.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 42

Use slide 42 to explain the configuration and deployment process of the Stateless Session bean.

Explain that Session beans can be configured in two ways. From EJB 3.0, the developers can use annotations to configure the beans or even use deployment descriptors to define the configuration information of the application. The use of deployment descriptor is optional.

The `ejb-jar.xml` is the deployment descriptor for the enterprise application. It is a declarative XML file which specifies the required configuration of the application. The deployment information determines how the EJB container manages the bean when deployed. This information includes the type of session in which the Session bean will establish, transaction type supported, and so on.

All EJB applications have `ejb-jar.xml` as the default deployment descriptor.

Tips:

Application deployment is not done manually always; there are several deployment tools which can be used to deploy the application. The deployment tools first try to access the deployment descriptor in the project archive file. Then, it proceeds further to deploy the application based on the declarations provided in the deployment descriptor.

Slide 43

Let us understand the default deployment descriptor for the Session bean.

□ Following code snippet demonstrates the deployment descriptor for the Session bean created earlier:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC
  '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
  'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>AdditionBean</ejb-name>
      <ejb-class>Beans.AdditionBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
    ...
  </ejb-jar>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 43

Use slide 43 to explain the deployment descriptor of the Session bean.

The initial header elements specify the information about the standards used in creating the deployment descriptor.

These Document Type Definitions (DTDs) describe the format to be followed while writing deployment descriptors (XML files). This format is useful to supply declarative data and assembly instructions for the archives.

The guidelines to be followed are specified through the guidelines defined at the link:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
```

The `<ejb-jar>` tag defines the actual configuration of the deployment descriptor.

The Session bean is configured through the `<session>` tag. The ejb to be used is specified through the `<ejb-name>`. The location of the bean class is specified through `<ejb-class>` tag. The type of the Session bean is specified through `<session-type>` tag and the type of the transaction is specified through `<transaction-type>`. The `<transaction-type>` element identifies the transaction management method applicable. In this case, the transaction management is defined through the container.

Tips:

There are two types of transactions in an EJB application, Container-managed and Bean-managed transactions. The default type of transaction is Container managed.

Slide 44

Let us understand the communication methods used by Session beans.

Communication with Session Beans

❑ Communication of the Session bean with client can be synchronous or asynchronous.

Synchronous communication <ul style="list-style-type: none"> • Client and bean are actively involved in the communication. • Reliable mode of communication. 	Asynchronous communication <ul style="list-style-type: none"> • Client sends a request and does not wait for response. • Used in case of long running operations.
---	--

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 44

Use slide 44 to explain different types of communication methods used by the Session beans in an application. Session beans can be accessed by the client either synchronously or asynchronously.

When accessed synchronously, the client sends a request to the application server and waits for response from the server. If the communication session has multiple request and response messages, the client and server respond to a message and wait for the consequent message without performing other operations. This type of communication is also known as synchronous communication where the client and server actively wait for responses from other entity. Synchronous communication reduces the amount of multitasking a server can perform due to waiting period.

In case of asynchronous communication, the client sends a request and does not actively wait for response from the server, it carries out other processes, while the request is processed by the server.

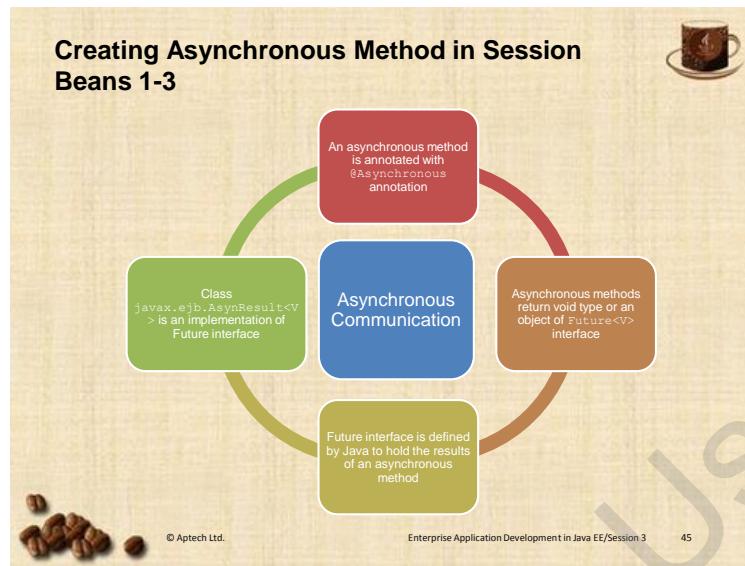
Then, explain how Session beans can be used. A Session bean is invoked by a client either through business interface view or no-interface view. The communication between the client and Session bean can be both synchronous and asynchronous.

Tips:

To get information on asynchronous communication, visit the following link:
<http://www.theserverside.com/news/1321142/Part-Three-New-Features-in-EJB-31>

Slide 45

Let us understand the process of creating asynchronous methods.



Use slide 45 to explain the process of creating asynchronous methods in the enterprise application.

Asynchronous methods in enterprise applications are annotated with `@Asynchronous`.

Asynchronous communication is used in case of long running operations. The client can perform other operations before the bean method execution is complete. Session beans which implement Web services cannot be asynchronous.

The return type of an asynchronous method should be of type `Future` interface. The `Future` interface can hold an object of any type that is to be returned by the asynchronous method. The Java EE provides this interface to hold objects which are expected as a result of executing an asynchronous method.

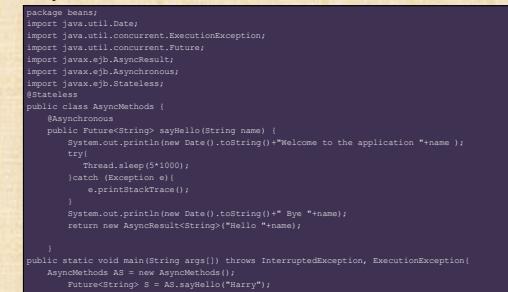
The `Future` interface provides methods to handle the return status of an asynchronous method.

The class `javax.ejb.AsyncResult<V>` is an implementation of `Future` interface. A method which returns an object of `Future` interface can throw application exceptions.

Slides 46 and 47

Let us understand asynchronous method invocation in Stateless Session bean.

Creating Asynchronous Method in Session Beans 2-3



```

package beans;
import java.util.Date;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import javax.ejb.AsyncResult;
import javax.ejb.Stateless;
import javax.ejb.Stateless;
public class AsyncMethods {
    @Asynchronous
    public Future<String> sayHello(String name) {
        System.out.println(new Date().toString() + "Welcome to the application "+name );
        try {
            Thread.sleep(5*1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(new Date().toString() + " Bye "+name);
        return new AsyncResult<String>("Hello "+name);
    }
}

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 46

Creating Asynchronous Method in Session Beans 3-3



Output

Java DB Database Process GlassFish Server AsynchronousApplication (run)

RUN:

Mon Jul 21 18:08:14 IST 2014 Welcome to the application Harry
Mon Jul 21 18:08:19 IST 2014 Bye Harry
In main method Hello Harry
BUILD SUCCESSFUL (total time: 5 seconds)

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 47

Use slides 46 and 47 to explain asynchronous method invocation in Stateless Session bean.

In the given code, `sayHello()` is an asynchronous method. The main method here is assumed to be the client which invokes the asynchronous method.

The code shows an asynchronous method which returns a `Future<String>` object. The `sayHello()` method in the code is annotated with `@Asynchronous` annotation and can be invoked asynchronously by the client. The `main()` method in the class can be seen as the client method invoking the asynchronous method. An asynchronous method can be invoked by another bean or an application client in an enterprise application. The asynchronous method returns an object of Future type.

The object is returned to the `main()` method. In order to retrieve the String data from the Future object, `get()` method is used as shown in the code.

Use slide 47 to demonstrate and show the execution of asynchronous method in the application.

Slide 48

Let us understand the Future interface.

Future Interface

- get() – This method waits if the method is not complete, if the method completes it returns an object of result type V.
- get(long TimeOut, TimeUnit unit) - This method waits for the result from the asynchronous method for the duration specified in the TimeOut parameter.
- cancel() – This method attempts to cancel an asynchronous method.
- isCancelled() – This method checks whether a method is cancelled or not and returns a boolean value.
- isDone() – This method checks whether a method is complete or not and returns a boolean value.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 48

Use slide 48 to explain the methods provided by the Future interface.

Following are the methods in Future interface:

- **get()** – This method waits if the method is not complete, if the method completes, it returns an object of result type V.
- **get(long TimeOut, TimeUnit unit)** - This method waits for the result from the asynchronous method for the duration specified in the **TimeOut** parameter, the timeout value is determined according to the **TimeUnit** objects.
- **cancel()** – This method attempts to cancel an asynchronous method, if it is not complete and returns a boolean value to specify whether the cancel operation was successful or not.
- **isCancelled()** – This method checks whether a method is cancelled or not and returns a **boolean** value.
- **isDone()** – This method checks whether a method is complete or not and returns a boolean value.

Additional References:

To learn more on the Future interface, visit the following links:

<http://java.dzone.com/articles/javauitconcurrentfuture>

<http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html>

Slide 49

Let us summarize the session.

Summary

- ❑ Stateless Session beans are used when the Session bean does not need to maintain the conversational state.
- ❑ Each Session bean is associated with one client.
- ❑ Container maintains a pool of Stateless Session beans.
- ❑ There are two stages in the lifecycle of the application – instantiated and removed.
- ❑ The container invokes the required callback methods for the Session bean.
- ❑ The enterprise application is deployed and configured through the deployment descriptor and annotations.
- ❑ ejb-jar.xml is the deployment descriptor for enterprise applications using EJB specification.
- ❑ Bean methods can be asynchronously invoked with the help of **Future<V>** interface.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 3 49

In slide 49, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Stateful Session beans, its life cycle, and its application which will be discussed in the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 4 – Stateful Session Beans

4.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

4.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the working of Stateful Session beans
- Explain the different elements of Stateful Session beans
- Describe the lifecycle of Stateful Session beans and associated callback methods
- Explain the Implementation of Stateful Session beans
- Explain the different types of clients accessing Stateful Session bean
- Explain exception handling in session beans

4.1.2 Teaching Skills

To teach this session successfully, you should be aware of the concept of Stateful Session beans and various aspects associated with it. You should also be aware of how Stateful Session beans are created and used in enterprise applications and be able to explain the characteristics associated with it, and have a clear understanding about the lifecycle of the Stateful Session bean.

You should describe the process of creating a Stateful Session Bean in a sample application and be able to demonstrate the application by executing it live in IDE.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- Explain the working of Stateful Session beans
- Explain the different elements of Stateful Session beans
- Describe the lifecycle of Stateful Session beans and associated callback methods
- Explain the Implementation of Stateful Session beans
- Explain the different types of clients accessing Stateful Session bean
- Explain exception handling in session beans

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 2

Tell them that they will be introduced to the concept of Stateful Session beans and various elements associated with it. They will also learn about different stages in the lifecycle of the Stateful Session Bean and associated callback methods.

Tell them that they will learn about the implementation of Stateful Session beans through NetBeans IDE. They will also learn about different types of clients that can access Stateful Session beans. Different types of exceptions that might arise when an application client communicates with the EJB and how developers can handle these exceptions in the application will be explained.

4.2 In-Class Explanations

Slides 3 to 5

Let us understand the purpose of Stateful Session beans.

Introduction 1-3



- ❑ Consider a situation where business processes invoked by the client needs the data to be maintained between the conversations over several requests.
- ❑ For example, while performing transactions on a bank account,
 - You may deposit as well as withdraw money from your account.

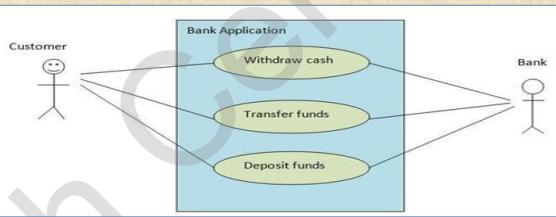


© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 3

Introduction 2-3



- ❑ Following figure shows a typical use case of a Bank application where application state has to be maintained:



- ❑ All processes need separate method invocations in the application and the state of the account data has to be maintained in order to perform the transactions.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 4

Introduction 3-3

- ❑ To process multiple requests, the enterprise bean specification has provided Stateful Session Beans.

❑ Stateful Session Bean:

- Stateful Session bean can be defined as a bean that services business processes spanning over multiple business method requests.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 5

Use slides 3 to 5 to explain the purpose of maintaining an application state in enterprise applications.

Consider the scenario where you fill in a Graphical User Interface (GUI) form. Whenever, you provide the data in the fields, the values are stored by the application. After you click Submit button, some more fields may be provided with the data based on the information you entered in the fields earlier. This means that the data entered in the fields maintain a conversation state of the client.

In this scenario, you can use Stateful Session beans that allow you to encapsulate the business logic and conversational state of the client on the server.

Compare this behavior with Stateless Session beans which do not hold the state of the data between the method invocations. This means after every method call, the container decides to either destroy the bean instance or pool the instance by clearing all the information related to the method invocation.

Use slide 4 to elaborate on the working of a bank application.

Tell them that the figure shown on slide presents a use case diagram where there are two users in the application context - Customer and Bank. A Customer can perform the operations on the account opened with the bank. The operations are represented by ellipses in the rectangle. The operations implemented in the bank application are Withdraw Cash, Transfer of funds, Deposit of Funds.

All the mentioned processes need separate method invocations in the application and the state of the account data has to be maintained in order to perform the transactions. Thus, to process multiple requests, the enterprise bean specification has provided Stateful Session beans.

Finally, use slide 5 to define the Stateful Session beans in the enterprise applications. Stateful Session bean can be defined as a bean that services business processes spanning over multiple business method requests.

Tips:

The Stateful Session beans acts as an agent for the clients. They process or task flow to accomplish a set of tasks. Stateful Session beans present a simplified interface that hides many independent operations performed by the application such as performing operations on the database and other beans from the clients.

Slides 6 and 7

Let us understand Stateful Session beans.

Stateful Session Bean 1-2

- ❑ Stateful Session bean maintains the conversational state of the application client with which it is associated.

An example of a Stateful session bean is a shopping cart on an e-commerce Web site. Each time you add a product to the cart or go to the next Web page, a new request is performed while retaining the state of the previous requests.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4

Stateful Session Bean 2-2

- ❑ A Stateful Session bean is dedicated to a single client.
- ❑ It retains the state of the client, until the Stateful Session bean instance is explicitly removed by the container or there is a timeout.
- ❑ Following figure shows the implementation of Stateful Session beans in a Bank application:

Client can access the EJB instance of the Stateful Session Bean through the EJB object.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4

Use slides 6 and 7 to explain the concept of Stateful Session beans and their usage in the application.

A Stateful Session bean maintains the conversational state of an application client onto the permanent storage. A Stateful Session bean is associated with a specific client. This means that the Stateful Session beans retain state on behalf of the client. In other words, if a Stateful Session bean's state is changed during a method invocation, the same state would be available to the same client upon the subsequent invocation.

An example of a Stateful session bean is a shopping cart on an e-commerce Web site. You can add multiple products to the shopping cart on different pages. Each time you add a product to the cart or go to the next Web page, a new request is performed while retaining the state of the previous requests.

Then, tell them that Stateful Session bean is dedicated to one client for the life of the bean instance. It retains the state on behalf of the client, until the Stateful Session bean instance is explicitly removed by the container or there is a timeout. The client can access the EJB instance of the Stateful Session Bean through the EJB object.

Using slide 7, explain them how Stateful Account Bean instances are accessed by the clients.

Slide 8

Let us understand the characteristics of Stateful Session bean.

Characteristics of a Stateful Session Bean

- Every instance of an application client is associated with a single instance of Stateful Session bean.
- Stateful Session beans can be activated or passivated.
- Stateful Session beans are transaction aware and short lived.
- Stateful Session beans are managed by EJB container.
- Stateful Session beans can access database, retrieve, and update data in the database.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 8

Use slide 8 to explain the characteristics of Stateful Session bean.

Explain the students that every instance of a Stateful Session bean is associated with only one client during the entire session. However, the remote clients can stop access the bean instance due to disconnecting from the network problem. In that case, the bean instance is left without a remote client. If the bean is not invoked in certain time duration, then the container decides to write the bean state on a disk and makes the bean instance ready to be passivated in the pool.

Discuss this point in comparison with Stateless Session bean. As Stateless Session bean do not have any state to passivate on the disk, so the container simply destroys the Stateless Session bean instance.

Then, explain the other characteristics of Stateful Session bean. They are transaction aware and short lived. Stateful Session beans can access the database, retrieve data from the database, and update data in the database as per application requirement.

In-Class Question:

After you finish explaining the code, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What decision is taken by the container when a Stateful Session bean instance is not invoked for longer duration by the client?

Answer:

If the bean is not invoked in certain time duration, then the container decides to write the bean state on a disk and makes the bean instance ready to be passivated in the pool.

Slides 9 to 12

Let us understand the process of passivation and activation of Stateful Session beans.

Stateful Session Bean Conversational State 1-4



When a Stateful Session bean is swapped out of the container its conversational state is written to the permanent storage.

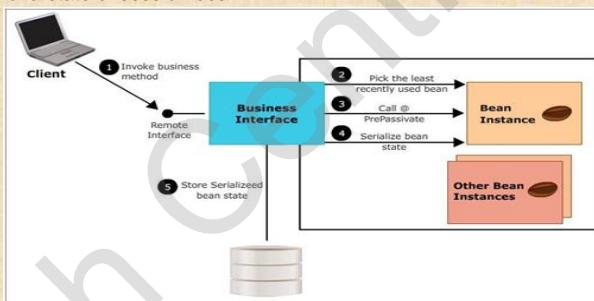
This process of writing the conversational state onto permanent storage and removing the Stateful Session bean from the container is known as passivation.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 9

Stateful Session Bean Conversational State 2-4



Following figure shows the process of storing the conversational state of a stateful session bean:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 10

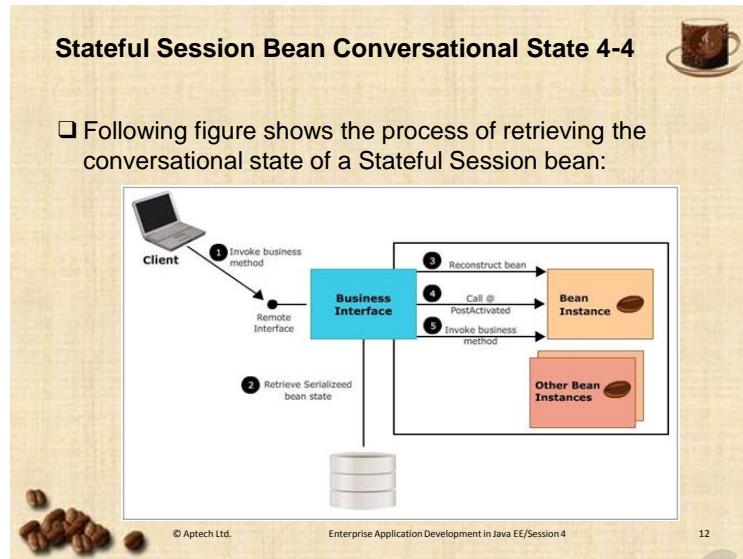
Stateful Session Bean Conversational State 3-4



To choose which Stateful Session Bean must be removed from the container, the container generally uses Least Recently Used (LRU) strategy.

When there is a request for the swapped out bean, then it is again brought back into the container, this process of bringing back the bean into container is known as activation.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 11



Use slides 9 to 12 to explain the process of passivation and activation of Stateful Session beans.

Discuss with them about conversational state.

Conversational state represents the values of the instance variables associated with the EJB object. When a client invokes a method on a bean instance, then the client is actually starting the conversation with the bean. The conversational state of the instance variables stored in the bean instance must be available to the same client in its next method request. Therefore, the container has to be careful while pooling the bean, as it dynamically assigns the bean to handle client method requests.

Ask the students what can be the benefit of achieving the pooling of the Stateful Session bean?

The effect of pooling conserves or saves the usage of resources such as memory, database connection, and socket connections. It also enhances the overall scalability of the system.

Using slide 9, explain that to limit the number of Stateful Session bean instances in the memory, the container can swap out a Stateful Session bean, saving its conversational state on a hard disk or some other storage such as a database. This process is called as passivation.

Using slide 10, explain the figure that shows the passivation of a Stateful Session bean. Tell them that the container informs the bean that it is about to passivate by calling the `PrePassivate` callback method.

Discuss with them on how the container decides which bean to passivate?

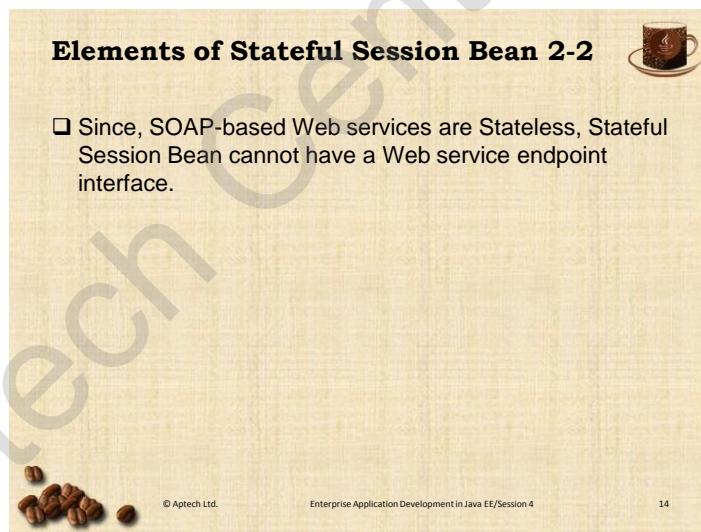
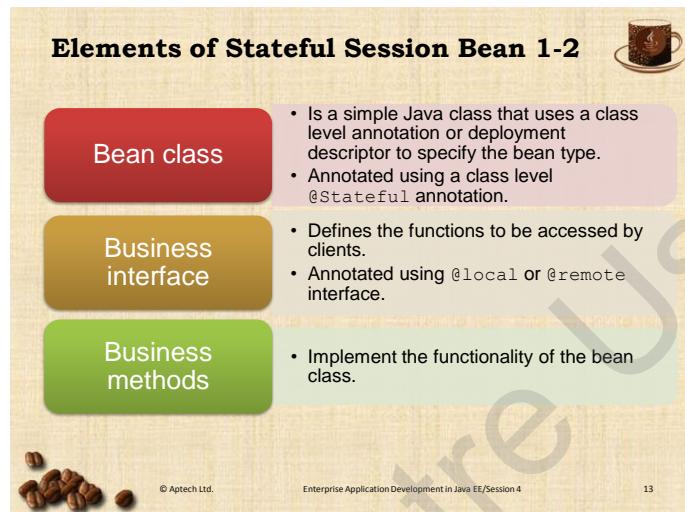
The technique of identifying which bean instance to passivate and which instance to activate depends on the application server used and its container. The container generally uses Least Recently Used (LRU) strategy. A Stateful Session Bean is said to be least recently used if it has not serviced any client requests for a long time.

Using slide 11, explain that when the same client returns back and invokes a method, the container takes the passivated state data and swaps in into some bean instance of the Stateful Session bean class taken from the pool. This process is called as activation.

Using slide 12, explain the figure that explains the process of activation of a Stateful Session bean. The passivated conversational state of the bean is read back into the memory and the bean state is reconstructed in the memory and assigned to the bean instance taken from the pool. The container calls the `PostActivation` callback method. The `PostActivation` callback method allows the bean to restore the open resources that were released during `PrePassivate` callback method.

Slides 13 and 14

Let us understand the components of the Stateful Session bean.



Use slides 13 and 14 to explain the elements of the Stateful Session bean.

A Stateful Session bean definition comprises a Bean class, its corresponding business interfaces and business methods of the class.

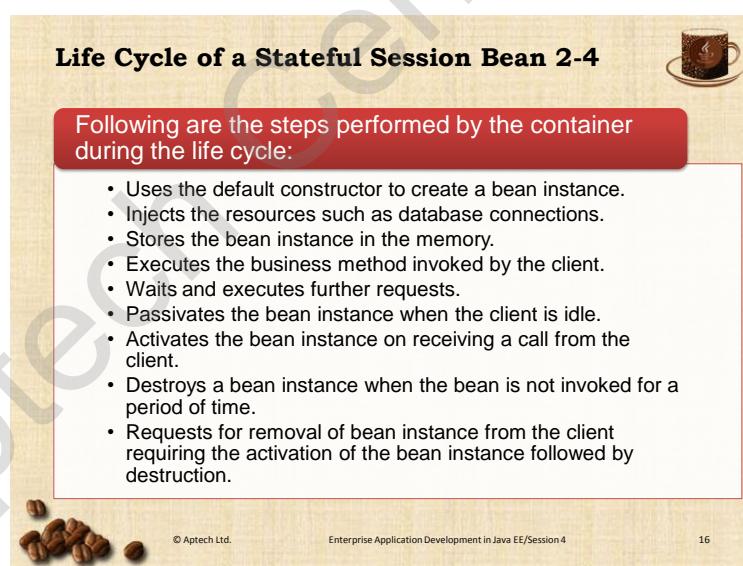
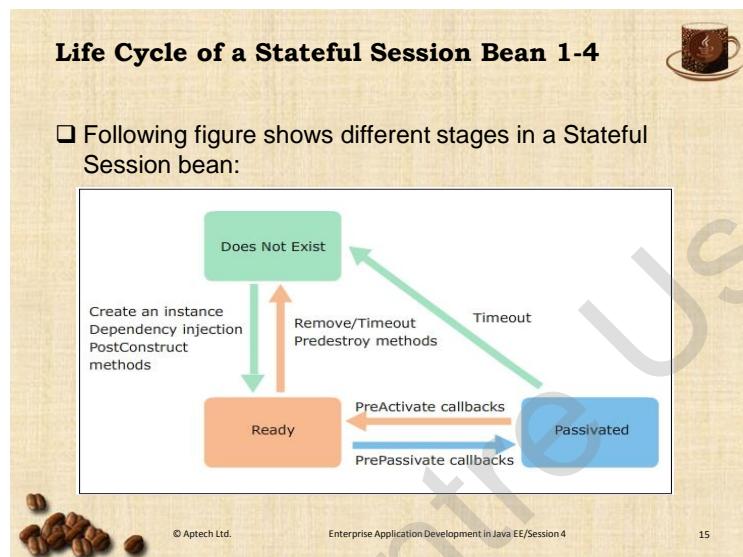
The Stateful Session bean is a simple Java class that uses a class level annotation or deployment descriptor to specify the bean type. The business interfaces of Stateful Session bean are similar to Stateless Session bean and are annotated using `@Local` and `@Remote` annotations. Since, SOAP-based Web services are Stateless, Stateful Session Bean cannot have a Web service endpoint interface. The business methods of a Bean class are methods defined in the class.

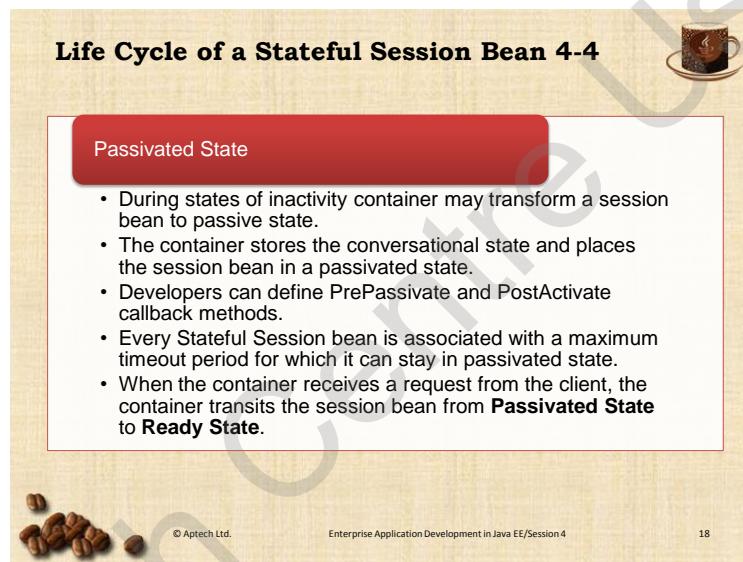
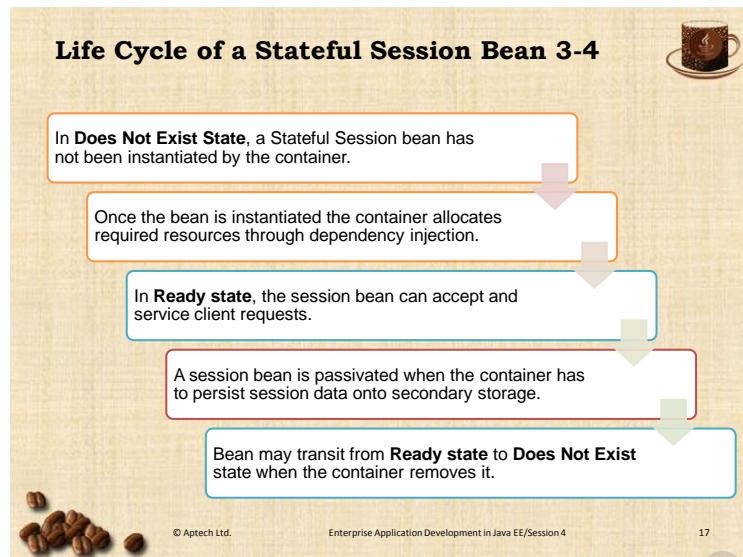
Tips:

Web services are stateless; they generally do not save the state of each client which is accessing the service. Therefore, Stateless Session beans are used with Web services. While defining Stateless Session beans in Web-enabled applications, a Web service endpoint is also defined. In case of Stateful Session beans, a Web service endpoint is not defined.

Slides 15 to 18

Let us understand the lifecycle of Stateful Session beans.





Use slides 15 to 18 to explain the lifecycle of Stateful Session beans.

Using slide 15, explain the figure that shows the various states in the lifecycle of a Stateful Session bean – Does Not Exist, Method-Ready, and Passivated.

Using slide 16, explain the steps that are performed by the container during the lifecycle of the Stateful Session bean.

Using slides 17 and 18, explain that initially the Stateful Session bean is in 'Does Not Exist' state. In this state, a Stateful Session bean has not been instantiated by the container. It does not hold any application resources in this state.

When the bean is instantiated it moves to Ready state. The bean is now in Ready state where it can accept client requests and service the requests. The bean's lifecycle begins when the client invokes the first method on the bean class. In this transition to the Method-Ready state, the `newInstance()` method is invoked by the container to create an instance of the bean class. The default constructor of the bean is invoked to perform necessary initialization.

The container injects any required context dependencies which are declared through annotations of XML deployment descriptor files. The container then, calls the `PostConstruct` call back method.

The `PostConstruct` call back method is an optional method and is invoked before the first business method invocation on the bean. This is at a point after which any dependency injection has been performed by the container.

The session bean may transit to Passivated State or Does Not Exist State from the Ready State. The session bean can transit to passivate, when the container has to persist session data onto secondary storage or when the client is waiting for some event to occur.

Before passivation, the container invokes the `PrePassivate` method, enabling the bean developer to clean up held resources, such as database connections, TCP/IP sockets, or any resources that cannot be transparently passivated using object serialization. Only certain object types can be serialized and passivated.

Discuss with them some of the situations when the bean is passivated are as follows:

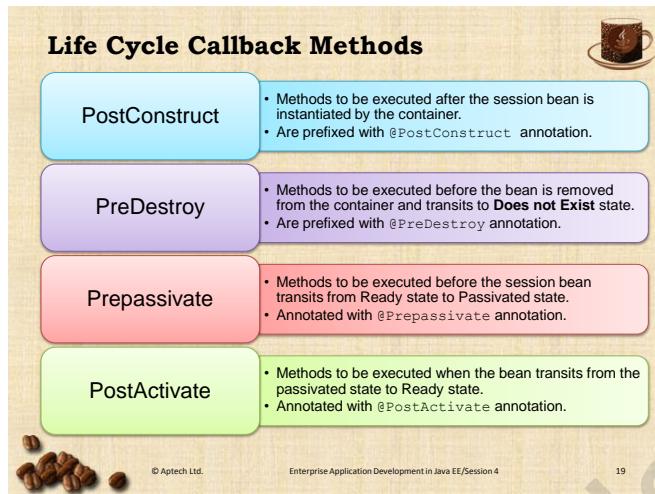
- Exceed idle timeout
- Exceed threshold for maximum number of instances or exceed absolute maximum number of instances
- Exceed threshold for maximum JVM memory consumption
- Shutdown application server instance

The Stateful Session bean moves from a Passivated state to Ready state when there is a request for the Stateful Session bean. When the bean is activated, the container executes the `PostActivate` methods defined for the bean. Tell the students that the definition of callback methods for a bean is optional.

A Stateful Session bean transits from a Ready state to Does Not Exist state when the bean is removed from the application. When a bean is removed corresponding `PreDestroy` callback methods are executed, if any of them are defined in the bean class. The bean's instance is then ready for garbage collection.

Slide 19

Let us understand the annotation used for defining the callback methods.



Use slide 19 to explain the annotations used to define the callback methods that can be defined with a Stateful Session bean.

Remember defining a callback method is not mandatory. The transition from one state to another state of the bean may or may not have a callback method. There are four variants of callback methods associated with Stateful Session bean based on their time of invocation:

- **PostConstruct** – These callback methods are executed after the session bean is instantiated by the container. They are used in both Stateful and Stateless Session beans. `PostConstruct` methods are prefixed with `@PostConstruct` annotation.
- **PreDestroy** – These callback methods are executed before the session bean is removed from the container and transits to **Does not Exist** state. These methods are also used by both Stateful and Stateless Session beans and are prefixed with `@PreDestroy` annotation.
- **PrePassivate** – These callback methods are used by the session bean, before the bean is passivated, that is the bean transits from Ready State to Passivated State. They perform all the required operations before passivating the bean such as persisting the session bean state to the secondary storage and so on. These methods are annotated with `@PrePassivate` annotation.
- **PostActivate** – These callback methods which are used to perform tasks after the container transits the session bean from the passivated state to active state. These methods are annotated with `@PostActivate` annotation.

Slide 20

Let us understand the programming rules for Stateful Session beans.

Programming Rules for Stateful Session Bean

Following are the rules to be implemented by a Stateful Session beans:

- Instances of Stateful Session bean should be of Java primitive data types or Serializable objects.
- Stateful Session bean class should define a method that would destroy the bean instance by the bean class using the `@Remove` annotation.
- Stateful Session bean have the `PostActivate` and `PrePassivate` lifecycle callback methods.
- The `PostActivate` method is invoked once the bean is brought back in the memory.
- The `PrePassivate` method is invoked before the bean instance is passivated.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 20

Use slide 20 to explain the programming rules for Stateful Session beans.

- Instances of Stateful Session bean should be of Java primitive data types or Serializable objects.
- Stateful Session bean class should define a method that would destroy the bean instance by the bean class using the `@Remove` annotation.
- Stateful Session bean have the `PostActivate` and `PrePassivate` lifecycle callback methods.

Slides 21 and 22

Let us understand the code for a Stateful Session bean.

Developing Stateful Session Bean 1-2



- ❑ Following code snippet demonstrates creating **ProductCatalogBean** class:

```
...
@Stateful
public class ProductCatalogBean implements ProductCatalogBeanRemote {
    List<String> products;
    public ProductCatalogBean() {
        products = new ArrayList<String>();
    }
    public void addProduct(String productName) {
        products.add(productName);
    }
    public List<String> getProducts() {
        return products;
    }
}
```

- ❑ Two business methods `addProduct()` and `getProducts()` are being created.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 21

Developing Stateful Session Bean 2-2



- ❑ Following code snippet shows the remote interface of a **ProductCatalogBean** class:

```
...
@Remote
public interface ProductCatalogBeanRemote {
    void addProduct(String productName);
    List getProducts();
}
```

- The annotation `@Remote` indicates that these methods can be accessed only by the remote clients.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 22

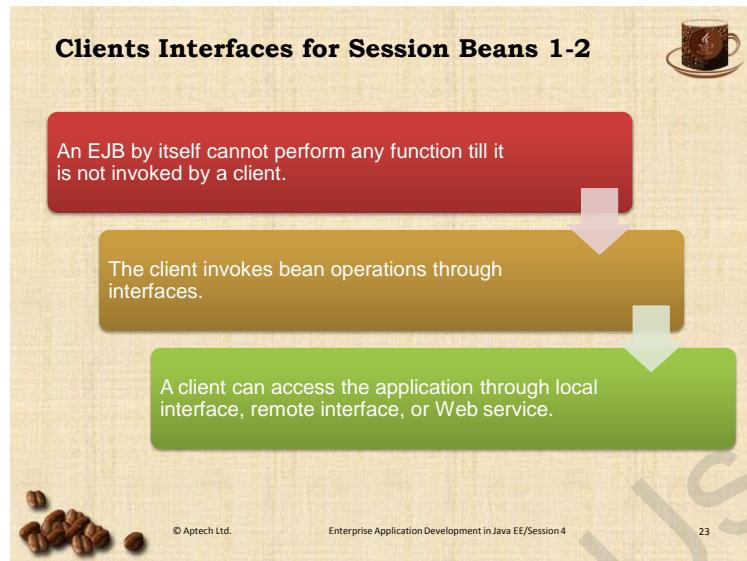
Use slides 21 and 22 to explain the code of Stateful Session bean.

A Stateful Session bean is created for the Shopping Cart application. The ProductCatalog is defined as an array of Product objects.

There are two methods defined in the Stateful Session bean – `addProduct()` and `getProducts()`. The Stateful Session bean has a remote interface which includes both the bean methods of the Stateful Session bean. This interface can be accessed by the remote clients of the application.

Slide 23

Let us understand the role of business interfaces associated with Stateful Session bean.



Use slide 23 to explain the role of business interfaces that are created for accessing Stateful Session beans.

Tell them that a client never gets access to the actual EJB. The EJB resides in the EJB container and only the container has the access to the bean. The client gets access only to the EJB object which contains the signature of all the methods that the client can call. The EJB object acts as a factory and distributor of EJB objects to the clients. Methods of a Stateful Session bean can be activated only through the interfaces.

When creating an enterprise application, the developer has to decide on how application clients can access the application. There are three types of interfaces that a Stateful Session bean can have – local interface, remote interface, and Web service endpoint.

Local interface is used by the local clients of the application. Remote clients use remote interface to access the Stateful Session bean. Web service is extended by the bean class for access by Web clients of the application.

Tips:

Local clients can also use no-interface view to access the bean. No-interface view comprises all the public methods of the bean class.

EJB 3.1 introduces the concept of a no-interface view, consisting of a variation of the Local view, which exposes all public methods of a bean class. Session Beans are not required to implement any interface anymore. The EJB container provides an implementation of a reference to a no-interface view, which allows the client to invoke any public method on the bean, and ensuring that transaction, security and interception behave as defined. Unlike local and remote views, where the reference consists of the respective local/remote business interface, the reference to a no-interface view is typed as the bean class.

Slide 24

Let us understand the factors involved in deciding the type of interface to be provided to the Stateful Session bean.

Clients Interfaces for Session Beans 2-2

Following are the factors which influence the decision of whether the clients should access the bean locally or remotely:

- Type of Service
- Relation among the bean components
- Location of components on the enterprise network
- Performance demands

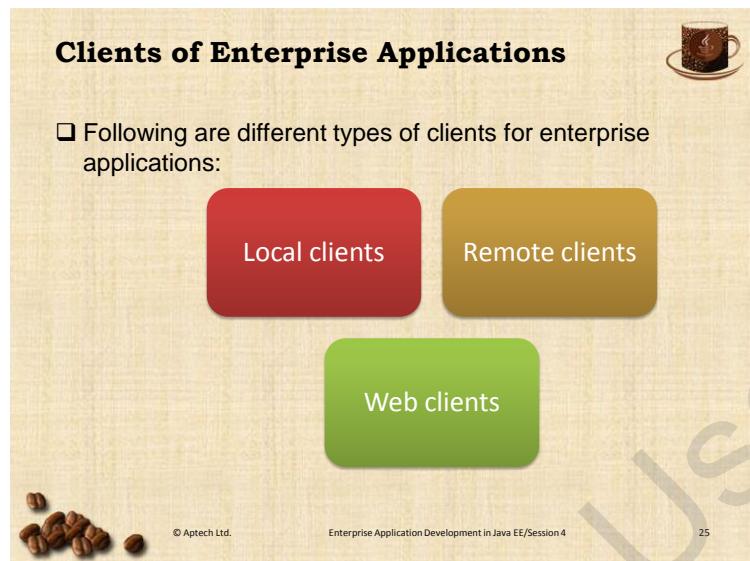
© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 24

Use slide 24 to explain the factors which define the type of interface to be provided to the clients of the application.

- **Type of Service** – Based on the type of the service provided by the application, an appropriate client and respective interface is defined for the bean.
- **Relation among the bean components** – Enterprise beans also take services from other beans. If the enterprise beans are tightly coupled, that is they have dependencies on each other it is a good design choice to create them as one logical unit. In such a scenario, the beans are provided with local access. If the bean components are loosely coupled then the beans can access each other remotely.
- **Location of components on the enterprise network** – A bean can be accessed either locally or remotely based on the distribution of application components over the enterprise network. If the remaining components of the application which have to access the enterprise beans are remotely located, then the bean component should be remotely accessed.
- **Performance demands** - Remotely located application components have performance degradation due to delays introduced by network traversal of data. The developer should consider these factors while deciding the type of access to the enterprise beans. Local access does not have these delays hence, the overall application performance is better.

Slide 25

Let us understand the different types of clients of an application.



Use slide 25 to introduce different types of clients for a Session bean.

There are three types of clients – Local clients, Remote clients and Web clients.

Local clients access the bean either through the local interface or through a no-interface view.

Remote clients access the bean through remote interface, the developer has to define the remote interface for the application. There can be multiple remote interfaces for a bean and the methods in a remote interface are a subset of the methods in the bean class.

Web clients access the bean through internet, the bean should define a Web service endpoint so that applications can be accessed over the Web.

Discuss with them on how an application running outside the Java EE server can access the EJB components.

Then, tell them that applications that run outside a Java EE environment must perform an explicit lookup for the EJB business objects using JNDI service. JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup.

Tips:

The Web service endpoint should enable the bean to be accessed through Internet protocols such as HTTP (Hyper Text Transfer Protocol). Session beans exposed through Web service endpoints are even accessible by non-Java client applications.

Slide 26

Let us understand more about local clients.

Local Client

- A local client and the session bean are located on the same JVM.
- A local client can be another enterprise bean or Web component of the application.
- Local clients can access the session beans either through no-interface or through local interface view.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 26

Use slide 26 to explain local clients of Session beans.

A local client and Session bean of an application are located on the same JVM. The access through local clients is quick as it does not introduce network traversal of application data.

Exposing an enterprise bean to a local client have a few drawbacks in terms of losing location transparency when local client and the enterprise bean are present in the same JVM.

Discuss with them which type of local clients can access the session beans?

A local client to the session bean can be another enterprise bean or Web component of the application.

A local interface need not be defined for local clients. They can access the Session bean through **no-interface view**. Now, tell them that the no-interface view of an enterprise bean is a local view. The enterprise bean class using no-interface view did not implement a business interface. This is the default interface for the session bean.

Annotations for defining a local business interface for the enterprise bean class. The business interface is decorated with `@Local` annotation.

For example, to annotate the `ProductCatalogBeanRemote` business interface, you can use the following code Snippet:

```
...
@Local
public interface ProductCatalogBeanLocal {
    void addProduct(String productName);
    List getProducts();
}
```

Or, you can also declare the local interface class in the session bean class.

Then, the bean implementation class can be specified with the interface name as shown:

```
@Local (ProductCatalogBeanLocal.class)
public class ProductCatalogBean implements ProductCatalogBeanLocal
{ ... }
```

Slide 27

Let us understand more about remote clients.

Remote Client 1-4



- Remote client may reside on a different JVM or a different physical machine.
- A remote client can be another enterprise bean residing in the same or different location.
- A remote client can also be a Web application, an applet, or a Java console application.
- Remote clients cannot access enterprise beans through no-interface view.
- Remote clients has to implement a business interface which is annotated with @Remote annotation.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 27



Use slide 27 to explain remote clients of an application.

A remote client to an EJB need not be located within the same JVM to access the methods of the bean. It can reside in a different JVM on another physical machine. The remote client does not care about the physical location of the bean that it accesses.

The remote client can be another enterprise bean residing in the same or different location, or it can also be a Web application, an applet, or a Java console application. The remote client can even be a non-Java program, such as a CORBA application written in C++.

Remote clients cannot access the bean through no-interface view and have to be annotated with @Remote. An enterprise bean allows access by remote clients through remote interface. The business and lifecycle methods of an enterprise bean are defined in the remote interface.

Slides 28 to 30

Let us understand how a remote client code accesses the Session bean.

Remote Client 2-4



Following code snippet shows a client accessing a shopping portal application:

```

public class StatefulClient{
    @EJB
    private static ProductCatalogBeanRemote
    productCatalogBean;
    public static void main(String[] args) {
        List PList = new ArrayList();
        productCatalogBean.addProduct("Laptop");
        productCatalogBean.addProduct("MobilePhone");
        productCatalogBean.addProduct("Personal Digital
Assistant");
    }
}

```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 28

Remote Client 3-4



```

productCatalogBean.getProducts();
// Iterate through all the elements of the collection
Iterator itr = PList.iterator();
while (itr.hasNext()) {
    String str = (String) itr.next();
    System.out.print(str + "\n");
}
System.out.println();
}
}

```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 29

Remote Client 4-4



Following figure shows the output of executing a remote client of the **ShoppingPortal** application:



The Output window shows the following build results:

```

run-single:
BUILD SUCCESSFUL (total time: 1 minute 31 seconds)

```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 30

Use slides 28 to 30 to demonstrate a remote client accessing the Session bean defined earlier.

The developer defines an array in the bean class which holds data for all the products in the Product catalog.

The client is using the remote interface `ProductCatalogBeanRemote` which has been injected through `@EJB` annotation. The client is browsing through the data stored in the array.

The `@EJB` annotation obtains the remote interface of the enterprise bean through dependency injection

The client adds three products to the catalog which in turn is persisted by the bean. The client then, invokes the `getProducts()` method to display a list of products in the application.

Use slide 30 to demonstrate the output of executing the client code.

Tips:

The client can obtain the reference of the enterprise beans either through `@EJB` annotation or JNDI lookup.

Slides 31 to 33

Let us understand the access of bean by the client through JNDI lookups

Accessing Through JNDI Lookups 1-3



The client can access the local or remote interface of an enterprise bean either through EJB objects or through lookup services such as JNDI.

JNDI is a naming and directory service provided by Java platform.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 31

Accessing Through JNDI Lookups 2-3



Following are the components which need to be set to access interfaces through JNDI lookup:

- JNDI initialization parameters
 - Includes configuring a JNDI driver.
 - Client must provide required properties to access the JNDI driver.
 - InitialContext provided by the JNDI API is used to set container specific properties.
- InitialContext class
 - Used to create an entry point into the naming system.
 - InitialContext provides the root to the hierarchy of the JNDI names in an application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 32

Accessing Through JNDI Lookups 3-3



Following code snippet demonstrates the usage of `InitialContext` object:

```
...
Context c = new InitialContext();
return (ProductCatalogBeanRemote)
c.lookup("java:global/ShoppingPortal/ShoppingPortal-
ejb/ProductCatalogBean!beans.ProductCatalogBeanRemote"
);
...
```

The code shows the usage of `InitialContext` object which returns the reference of `ProductCatalogBeanRemote` interface.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 33

Use slides 31 to 33 to explain how the client accesses the bean through JNDI lookups.

Clients use JNDI lookup services through annotations. JNDI is a naming service provided by Java EE platform to provide a uniform mechanism to access all application components.

Use slide 32 to explain elements to be set when the remote interface is accessed through JNDI lookup.

The first task would be setting the JNDI initialization parameters, which includes configuring the JNDI driver. To access the JNDI service, the client must provide the manual properties for accessing the container specific JNDI driver in the client program.

All the components in the JNDI are arranged in a hierarchy, the `InitialContext` of the application provides the entry point to this hierarchy. It can also be termed as the root to the hierarchy of the application components.

Tips:

It is not a preferred way to manually code the JNDI properties. The developers should use the default initial context provided by the JNDI API.

Use slide 33 to demonstrate the code for defining the `InitialContext` of the application.

The initial context of the application is set, where through the `lookup()` method the remote interface of the bean is returned.

Slide 34

Let us understand how the JNDI is configured.

Configuring JNDI 1-2

- A lookup operation in the JNDI namespace is performed through `lookup()` method.
- The `lookup()` method uses the deployment descriptor to obtain the JNDI name and object binding.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 34

Use slide 34 to explain the configuration of JNDI. The `lookup()` method of the JNDI API is used to obtain the reference of the application component.

The mapping of the JNDI name and the corresponding component is present in the deployment descriptor of the application. The `lookup()` method uses this mapping information to obtain a reference of the component.

Slide 35

Let us understand the mapping in a deployment descriptor.

Configuring JNDI 2-2

- Following code snippet shows a sample deployment descriptor with JNDI name bindings defined:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC "-//GlassFish.org//DTD
GlassFish Application Server 3.1 EJB 3.1//EN"
"http://glassfish.org/dtds/glassfish-ejb-jar_3_1-1.dtd">
<glassfish-ejb-jar>
    <enterprise-beans>
        <ejb>
            <ejb-name>ProductCatalogBean</ejb-name>
            <jndi-name>pro</jndi-name>
        </ejb>
    </enterprise-beans>
</glassfish-ejb-jar>
```

- In Code Snippet, the `ProductCatalogBean` is bound to the JNDI name 'pro'.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 35

Use slide 35 to show the application component and its corresponding JNDI name mapping.

In the given code the EJB `ProductCatalogBean` identified through `<ejb-name>` element is mapped onto the JNDI name `pro`.

Slide 36

Let us understand resource injection in enterprise beans.

Injecting Resources into Enterprise Beans

- ❑ Resource injection enables developers to inject resources such as databases, connectors, and so on into container-managed components.
- ❑ Resources to be injected must be defined in the JNDI namespace.
- ❑ Resources are referred through `@javax.annotation.Resource` annotation.
- ❑ `@Resource` annotation has six attributes – `name`, `type`, `authenticationType`, `shareable`, `mappedName`, and `description`.
- ❑ Resources can be injected at the following levels in the code:
 - Field level
 - Method level
 - Class level

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 36

Use slide 36 to explain how resources can be injected into enterprise beans. Resources in an enterprise application are components such as Connections, Databases, and so on.

Java EE provides `@Resource` annotation to inject these resources into the application. The annotation can inject resources at different levels – method level, field level, and class level. When a resource is injected at method level then, the resource can be used throughout the execution of the method. When a resource is injected at field level, the annotation is prefixed before the variable declaration and the resource can be used with the variable.

The resource injected at the class level can be used by the objects of the class during the execution of class methods.

The `@Resource` annotation has six attributes –

- `name` field has the JNDI name of the resource being accessed.
- `type` implies the type of the resource.
- `authenticationType` refers to the method of authentication used by the resource, if any.
- `shareable` implies whether the resource can be shared with other components.
- `mappedName` implies any system specific name onto which the resource has to be mapped.
- `Description` describes the resource.

Slide 37

Let us understand exception handling in Session beans.

Exception Handling in Session Beans

- ❑ Exception is an unusual condition/control flow in an application.
- ❑ Exceptions might be caused by user errors or programming errors.
- ❑ Exception handling is done through:
 - `try-catch` block
 - `throws` statement

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 37

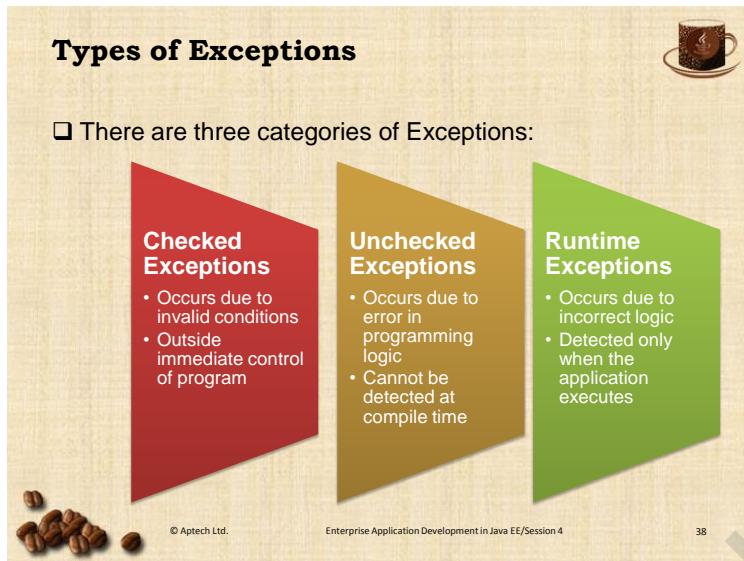
Use slide 37 to explain exception handling in Session beans. Exception handling is one of those features of Java EE, which make the application robust. Exception handling enables the application to gracefully degrade on encountering unwarranted conditions. An exception is an event, which occurs during the execution of a program which disrupts the normal flow of program's instructions.

Java provides try-catch blocks and throws statements to handle exceptions in the application.

In case of a `try-catch` block the code which is likely to raise an exception is enclosed in the `try` block and the `catch` block comprises code to be executed to handle the exception which occurred in the `try` block.

Slide 38

Let us understand the different types of exceptions.



Use slide 38 to explain different types of exceptions.

There are three types of exceptions which might occur in an enterprise application.

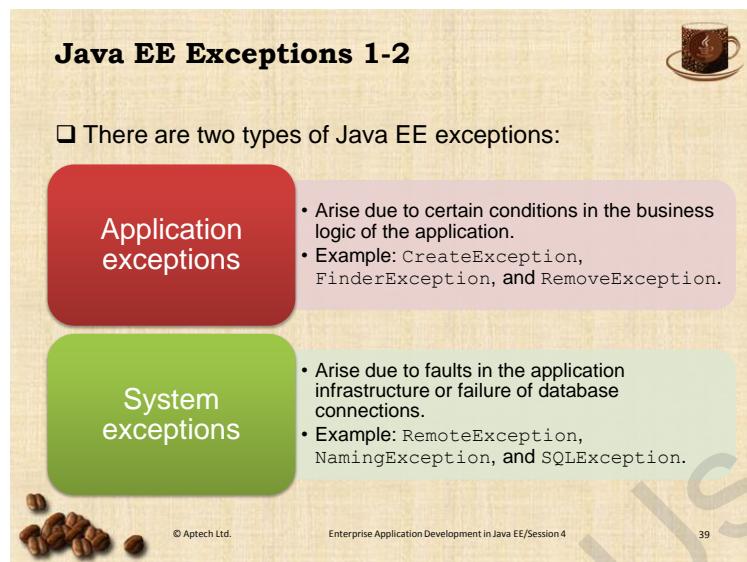
Checked exceptions are exceptions which occur due to some unforeseen conditions which might occur during application execution.

Unchecked exceptions are those which occur due to error in programming logic, but cannot be detected during compile time.

Runtime exceptions occur during the application execution.

Slide 39

Let us understand the exceptions specific to enterprise applications.



Java EE Exceptions 1-2

□ There are two types of Java EE exceptions:

- Application exceptions
 - Arise due to certain conditions in the business logic of the application.
 - Example: CreateException, FinderException, and RemoveException.
- System exceptions
 - Arise due to faults in the application infrastructure or failure of database connections.
 - Example: RemoteException, NamingException, and SQLException.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 39

Use slide 39 to explain different types of Java EE exceptions. Java EE exceptions are those which may occur in enterprise applications due to their huge scale, due to the distributed nature of the application and so on.

There are two categories of Java EE exceptions, they are application exceptions and system exceptions.

Application exceptions arise due to certain conditions in the business logic of the application.

System exceptions arise due to faults in the application infrastructure or failure of database connections.

In-Class Question:

After you finish explaining exceptions specific to enterprise applications, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are checked exceptions?

Answer:

A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer.

Slide 40

Let us understand different types of exceptions which may occur in enterprise applications.

Java EE Exceptions 2-2



Following table shows different exceptions that might occur in EJB applications:

Exception	Description
CreateException	Raised when the instantiation of an object or bean fails.
FinderException	Raised when the application is unable to find an object it is looking up for.
RemoveException	Raised when the application cannot remove an instance of bean, cannot be removed from the container.
RemoteException	Raised when there is a network failure and the client cannot access a remote object.
NamingException	Raised when the JNDI name could not be resolved to the object which is looked up for.
SQLException	Raised when the bean could not get response from the application database.
EJBException	Raised when the bean could not respond to the application clients appropriately.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 40

Use slide 40 to explain different exceptions which might occur in enterprise applications.

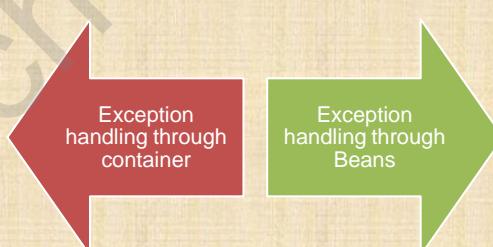
Slide 41

Let us understand the exception handling methods in the application.

Exception Handling in Enterprise Applications



Exceptions in enterprise applications are handled in two ways:



Exception handling through container
Exception handling through Beans

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 41

Use slide 41 to explain the methods of enterprise applications.

Exceptions in enterprise applications can be handled either through container or through beans.

Container-managed exception handling is the default method of exception handling. In case of bean-managed transactions, the developer has to explicitly define the exception handling mechanism.

Slide 42

Let us understand exception handling through containers.

Handling Exceptions Through Containers

Container handles application exceptions by returning it to caller and executing the exception handling code.

Container performs the following operations to perform system exceptions:

- Container logs the system exception
- Deallocates the allocated resources and performs clean up operation
- Removes bean instance from the memory
- The calling method is informed about the exception thrown

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 42

Use slide 42 to explain exception handling through containers. The developer need not define any steps of exception handling. Whenever an exception occurs, the container logs the exception and returns to the method where the exception has occurred.

The container deallocates the memory allocated for the bean and removes the bean instance from the memory.

Slides 43 and 44

Let us understand bean managed exception handling.

Handling Exceptions Through Beans 1-3



Beans handle exceptions by writing explicit code.
Following code snippet demonstrates usage of exceptions in enterprise applications:

```
public class ExceptionDemo {
    public static void main(String[] args) throws
        FileNotFoundException, IOException {
        try{
            testException(-5);
            testException(-10);
        }catch(FileNotFoundException e){
            e.printStackTrace();
        }catch(IOException e){
            e.printStackTrace();
        }finally{
            System.out.println("Releasing resources");
        }
        testException(15);
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 43

Handling Exceptions Through Beans 2-3



```
public static void testException(int i) throws
FileNotFoundException, IOException{
    if(i < 0){
        FileNotFoundException myException = new
        FileNotFoundException("Negative Integer "+i);
        throw myException;
    }else if(i > 10){
        throw new IOException("Only supported for index 0
        to 10");
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 44

Use slides 43 and 44 to explain code which uses bean managed exception handling. In case of bean managed exception handling, the developer uses try-catch blocks and throws statements to handle exceptions.

The try-catch block has a method invocation which throws an exception. The code in the `testException()` method throws exception explicitly.

The `finally` block in the code deallocates all the resources allocated.

In-Class Question:

After you finish explaining the code, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Why is finally block of statements used in the application?

Answer:

A finally block of statements is used to deallocate resources of the class.

Slide 45

Let us look at the output of the program with bean managed exception handling.

Handling Exceptions Through Beans 3-3

Following figure shows the output of the exception handling code:

```
Output - BankApplication-ejb (run) ✘
RUN:
java.io.FileNotFoundException: Negative Integer -5
Releasing resources
    at beans.ExceptionDemo.testException(ExceptionDemo.java:28)
    at beans.ExceptionDemo.main(ExceptionDemo.java:10)
Exception in thread "main" java.io.IOException: Only supported for index 0 to 10
    at beans.ExceptionDemo.testException(ExceptionDemo.java:34)
    at beans.ExceptionDemo.main(ExceptionDemo.java:19)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 45

Use slide 45 to explain the output of a program with bean managed exception handling. The output shows the exceptions explicitly thrown in the `testException()` method.

Slide 46

Let us understand exception logging.

Exception Logging 1-4

- ❑ Exceptions are logged by applications for further analysis.
- ❑ Java provides `java.util.logging` package to implement logging.
- ❑ Applications implement logging to:
 - Diagnose any problem in the application
 - Trace the application functionality
 - Root cause the problem in case of a system crash

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 46

Use slide 46 to explain exception logging in applications.

Enterprise applications use logging of operations and exception logging to keep track of the operation of application. Logging operations are essential for recovery of application in case of application failure.

Exceptions can be logged at multiple levels – Component-level logging, middle level logging, and top level logging.

Java EE provides for logging through `java.util.logging` package.

Component level logging of the exceptions refer to exception logging at each component of the application. Whenever a component throws an exception it also makes an entry into the exception log. This mechanism of exception logging requires logging code to be written for each component.

In case of middle level exception logging, the exception log is maintained in a location common to different components where sufficient information about all the components is available such as a container. Middle level logging does not have information about the entire application but a subset of application components.

Top level logging of exceptions has information about all the exceptions thrown in the application. Such exception logs are useful to assess the performance of the entire application.

Slides 47 and 48

Let us look at a sample exception logging code.

Exception Logging 2-4

□ Following code snippet demonstrates exception logging in enterprise applications:

```
private com.card.CardValidationRemote
lookupCardValidationBean() {
try {
javax.naming.Context c = new javax.naming.InitialContext();
Object remote = c.lookup(...);
com.card.CardValidationRemoteHome rv = ... return
rv.create();
} catch(javax.naming.NamingException ne) {

java.util.logging.Logger.getLogger(getClass().getName()).lo
g java.util.logging.Level.SEVERE,"exception caught" ,ne);
throw new RuntimeException(ne);
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 47

Exception Logging 3-4

```
} catch(javax.ejb.CreateException ce) {
java.util.logging.Logger.getLogger(getClass().getName()
()).log(java.util.logging.Level.SEVERE,"exception caught"
,ce);
throw new RuntimeException(ce);
} catch(java.rmi.RemoteException re) {

java.util.logging.Logger.getLogger(getClass().getName()).log
(java.util.logging.Level.SEVERE,"exception caught" ,re);
throw new RuntimeException(re);
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 48

Use slides 47 and 48 to demonstrate code which implements logging at different levels.

The code uses `java.util.logging` package to implement logging. The statements highlighted in yellow perform the required logging operations.

Slide 49

Let us understand the various levels of exception logging.

Exception Logging 4-4

`java.util.logging` package provides various levels of logging in the application.

Following are predefined level constants:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 49

Use slide 49 to explain different levels of exception logging.

The Level class defines standard logging levels to control logging output.

Developers normally use the predefined Level constants such as Level.SEVERE.

The levels in descending order are:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

Slide 50

Let us summarize the session.

Summary

- ❑ Stateful Session beans store the conversational state of the session.
- ❑ Each Stateful Session bean has a unique identity and is associated with a single client.
- ❑ There are three states in the life cycle of a Stateful Session bean – Does not Exist, Activated, and Passivated.
- ❑ There are four categories of life cycle callback methods – PostConstruct, PrePassivate, PostActivate, and PreDestroy.
- ❑ Stateful Session beans can be accessed through both local and remote interface.
- ❑ Stateful Session beans can be activated through local, remote, and Web service clients.
- ❑ Application and system exceptions are two types of exceptions in an enterprise application according to EJB specification.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 4 50

Using slide 50, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

4.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Singleton Session beans that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 5 – Singleton Session Bean

5.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

5.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the need of Singleton Session bean
- Describe the characteristics of Singleton Session bean
- Describe the various stages in the lifecycle of Singleton Session bean
- Explain Singleton Session bean specification and its initialization strategies
- Describe how to achieve concurrency access to Singleton Session bean
- Explain container-managed concurrent access to Singleton Session bean
- Explain bean-managed concurrent access to Singleton Session bean
- Explain the mechanism to configure access time out in concurrency
- Explain how to implement Singleton Session bean in an enterprise application

5.1.2 Teaching Skills

To teach this session successfully, you should be aware of the concept of Session beans in enterprise applications. You should be aware of the different types of Session beans provided by Java EE platform and their utility and how developers implement them.

You should be able to explain Singleton Session beans, their characteristics, and implementation of a Singleton Session bean through NetBeans IDE.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❑ Explain the need of Singleton Session bean
- ❑ Describe the characteristics of Singleton Session bean
- ❑ Describe the various stages in the lifecycle of Singleton Session bean
- ❑ Explain Singleton Session bean specification and its initialization strategies
- ❑ Describe how to achieve concurrency access to Singleton Session bean
- ❑ Explain container-managed concurrent access to Singleton Session bean
- ❑ Explain bean-managed concurrent access to Singleton Session bean
- ❑ Explain the mechanism to configure access time out in concurrency
- ❑ Explain how to implement Singleton Session bean in an enterprise application

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 2

Tell them that they will be introduced to Singleton Session beans, their characteristics, their lifecycle and other aspects associated with it. Singleton Session beans are generally used for tasks such as application initialization which apply to the entire application. There is only one instance of a Singleton Session bean in the application.

While working with a Singleton Session bean, a developer also has to implement concurrency control mechanisms as multiple clients may try to access the instance of a Singleton Session bean. The container also provides a concurrency control mechanisms.

5.2 In-Class Explanations

Slides 3 and 4

Let us understand the need of Singleton Session bean.

Introduction 1-2

The diagram illustrates the access mechanism for session beans. On the left, there are three rectangular boxes labeled "Request". Dotted arrows connect each request to a central vertical rectangle labeled "EJB Container". Inside the "EJB Container" is a dashed rectangular box labeled "Instance Pool". Inside the "Instance Pool", there are five small circles arranged in two columns: two in the top row and three in the bottom row. This visualizes how multiple requests are handled by a single instance of the bean.

□ In both, the Stateless Session bean and Stateful Session bean model:

- Only single request can access the bean instance at any time. This means these beans are thread-safe, as each request is represented by the invocation of a single thread.

□ Following figure demonstrates how session beans are accessed from a pool of bean instances:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 3

Introduction 2-2

A red question mark icon is positioned to the left of a text box. The text box contains the following information:

EJB 3.1 specification have introduced
Singleton Session bean which provides
concurrent access to the clients.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 4

Use slides 3 and 4 to explain the need of a Singleton Session bean.

Discuss with the students about the requirement for creating Stateless and Stateful Session beans in the enterprise applications.

Brief out the explanation saying that Stateless Session bean offer better scalability for the applications, as few instances can support large number of clients. However, Stateful Session beans are associated with specific clients, hence, each new clients request, creates a new instance of the Stateful Session bean.

Then, explain the figure shown on slide 3 to explain how the session bean instances are pooled by the container. Tell them that the Stateless and Stateful Session beans are thread-safe, as each request is represented by the invocation of a single thread.

Now, discuss a scenario where a concurrent access to the session bean is required. For example, a global stock market listing or daily price list which will be same or static for every user accessing the bean. This concurrent access to the bean will repeat the query execution for the same data for multiple requests.

Thus, to provide concurrent access to the bean without instantiation it for every client requests, EJB 3.1 specification has introduced Singleton Session beans.

Singleton Session bean is different from Stateless and Stateful Session beans because it can be accessed by multiple clients of the application simultaneously. Stateless and Stateful Session beans cannot be accessed by multiple clients simultaneously.

Thus, when the developer wants only one instance of the bean is created, regardless of the number of clients accessing the resource, then Singleton Session bean component can be developed.

Tips:

Since, a Singleton Session bean is accessed by multiple clients simultaneously, container has to implement concurrency control techniques on the Singleton Session bean to maintain the integrity of the data stored in the bean.

Slides 5 and 6

Let us understand Singleton Session bean.

Singleton Session Bean 1-2



A Singleton Session bean:

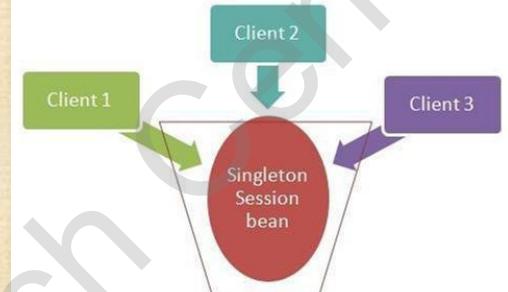
- Is one which gets instantiated only once for every application.
- Exists during the entire lifecycle of an application.
- One or more clients can simultaneously access the same bean instance at the same time.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 5

Singleton Session Bean 2-2



Following figure demonstrates how a single bean instance can be accessed by multiple clients:



The diagram illustrates the architecture of a Singleton Session bean. At the center is a red oval labeled "Singleton Session bean". Three arrows point from three separate colored boxes labeled "Client 1" (green), "Client 2" (teal), and "Client 3" (purple) towards the central bean instance. The background of the slide features a faint watermark of coffee beans.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 6

Use slides 5 and 6 to explain Singleton Session bean.

Singleton Session bean is an enterprise bean similar to that of Stateless and Stateful Session beans.

In an enterprise application, there is only one instance of a Singleton Session bean, as it is instantiated only once and is retained throughout the life of the application. It supports concurrent access to the clients. This means one or more clients can simultaneously access the same bean instance at the same time.

They are deployed in the application container similar to other enterprise beans. During its lifecycle they are always in the activated state. This means that as there is a single instance of the Singleton Session bean, it is alive throughout the application and cannot be passivated in the pool like Stateful Session beans.

Then, explain the figure shown on slide 6 for accessing the Singleton Session beans. Tell them that a Singleton Session bean instance lives for the entire lifecycle of the application. It maintains the state between the client's methods invocation, however, the state is not survived if the container crashes or application is shutdown.

Developers use a Singleton Session bean to implement tasks such as initializing the application and cleaning up before the application is shut down, application logging and auditing of application performance, and so on.

Tips:

The application server function may be distributed over multiple servers and therefore, the application container is also scaled over multiple servers. In such a situation, every instance of application container on different server machines will have an instance of the Singleton Session bean.

In-Class Question:

After you finish with the explanation on Singleton Session beans, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are the circumstances in which Singleton Session bean is appropriate?

Answer:

- a. When the same state of the bean needs to be shared across the application.
- b. When a single enterprise bean needs to be accessed by multiple threads concurrently.
- c. When application needs to perform startup and shutdown tasks.

Slide 7

Let us understand the characteristics of a Singleton Session bean.

Characteristics of a Singleton Session Bean

- It is shared by all requests.
- It is similar to Stateless Session bean.
- Like Stateless Session beans, the conversational state of Singleton Session beans cannot be persisted onto permanent storage.
- Its instance cannot be passivated and must be thread-safe.
- It has less memory footprint as compared to other session beans.
- Container is responsible for deciding when to initialize a Singleton bean instance.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 7

Use slide 7 to explain the characteristics of a Singleton Session bean.

Singleton session beans offer similar functionality to Stateless Session beans. However, they differ from them in that there is only one singleton session bean per application, as opposed to a pool of Stateless Session beans, any of which may respond to a client request. Tell them that like Stateless Session beans, Singleton Session beans can implement Web service endpoints.

Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

As the Singleton Session bean does not maintain the conversational state of the bean, it occupies less memory in the container deployed on the application server.

Then, explain them the due to concurrent invocations, the Singleton Session bean instances must be designed as thread-safe. The mechanisms such as locking or synchronization ensures thread-safety by blocking the threads coming from for the new client requests.

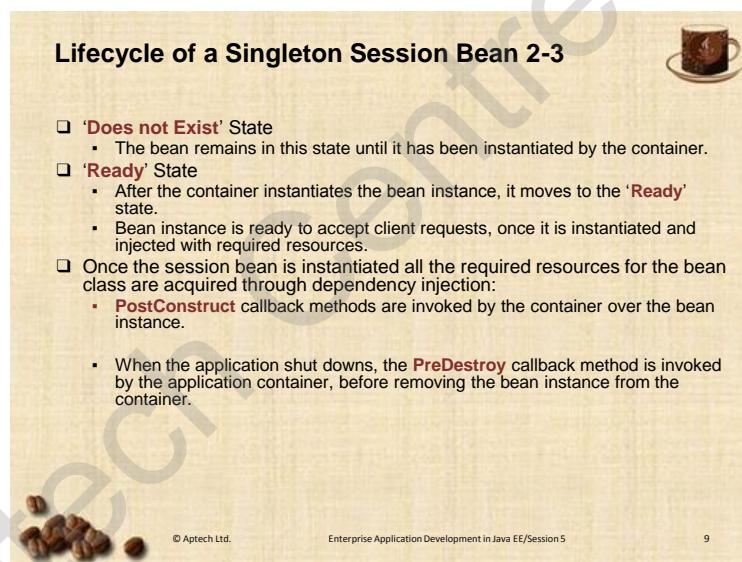
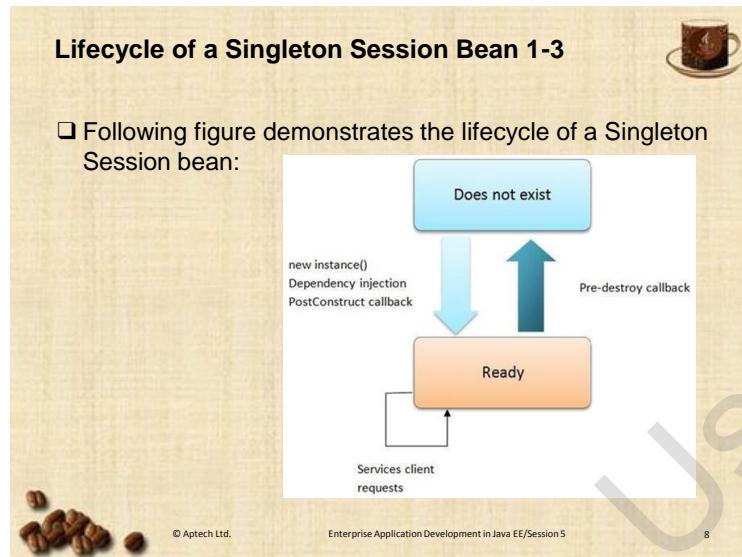
The Singleton Session bean need not be instantiated at the beginning of the application and is retained till the end of the application life. The container can decide based on application requirement when to instantiate a Singleton Session bean.

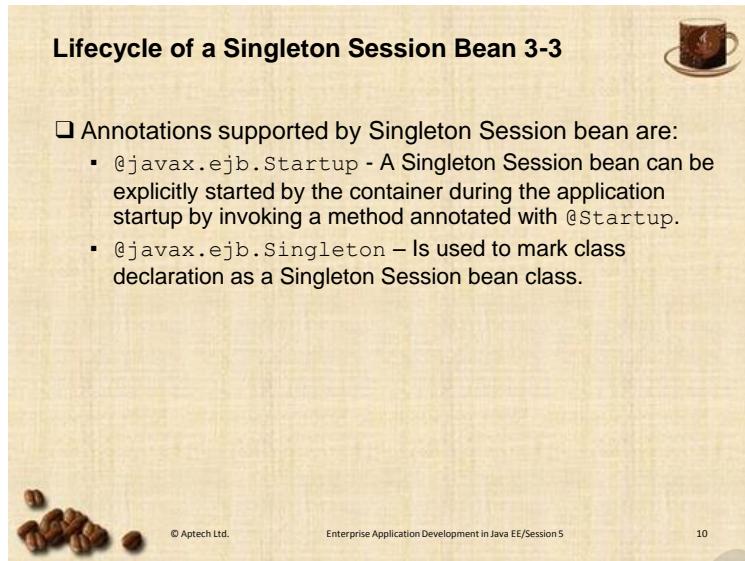
Tips:

Applications that use a Singleton Session bean may specify that the Singleton should be instantiated upon application startup, which allows the Singleton to perform initialization tasks for the application. The Singleton bean may perform cleanup tasks on application shutdown as well, because the Singleton Session bean will operate throughout the lifecycle of the application.

Slides 8 to 10

Let us understand the lifecycle of a Singleton Session bean.





Lifecycle of a Singleton Session Bean 3-3

❑ Annotations supported by Singleton Session bean are:

- `@javax.ejb.Startup` - A Singleton Session bean can be explicitly started by the container during the application startup by invoking a method annotated with `@Startup`.
- `@javax.ejb.Singleton` – Is used to mark class declaration as a Singleton Session bean class.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 10

Use slides 8 to 10 to explain the lifecycle of a Singleton Session bean.

Explain the lifecycle of a Singleton Session bean figure shown on slide 8.

Similar to a Stateless Session bean, there are two states in the lifecycle of a Singleton Session bean – **Does Not Exist** and **Method Ready** state. There are two types of callback methods associated with the lifecycle of the Singleton Session bean – **PostConstruct** methods and **PreDestroy** methods.

A Singleton Session bean can be explicitly started by the container during the application startup by invoking a method annotated with `@javax.ejb.Startup`. The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if it exists. The singleton session bean is now ready to have its business methods invoked by the client.

The Singleton Session bean comes into existence when the container instantiates it. Like a Stateless Session bean, it is ready to accept client requests, once it is instantiated.

At the end of the lifecycle, the EJB container calls the method annotated `@PreDestroy`, if it exists. The singleton session bean is now ready for garbage collection.

Use slide 10 to explain the annotations associated with Singleton Session bean.

- `@Startup` annotation indicates the actions to be performed by the container during application startup. The annotation specifies operations such as initializing a Singleton Session bean. `@Startup` annotation can also be used for initializing other aspects of the application.
- `@Singleton` is used to mark the class as a Singleton Session bean. An instance of the class annotated with `@Singleton` is created by the container as a Singleton Session bean.

It has three attributes associated with it:

- name – name of the bean or the bean's class name.
- mappedName – An application specific name(e.g. global JNDI name) that this Session bean should be mapped to.
- description – A string describing the Singleton Session bean.

Singleton Session bean also has callback methods associated with it. Hence, the annotations `@PostConstruct` and `@PreDestroy` can also be used along with the Singleton Session bean.

Tips:

There are two techniques of initialization used in enterprise applications – eager initialization and lazy initialization. In case of eager initialization, the instance variables and variables pertaining to the Singleton Session bean are initialized as soon as the application starts up. In case of lazy initialization the initialization is done as per requirement. Eager initialization is more resource intensive as compared to lazy initialization.

In-Class Question:

After you finish explaining the lifecycle of Singleton Session Bean, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which annotation indicates the actions to be performed by the container during application startup?

Answer:

`@javax.ejb.Startup`

Slide 11

Let us understand the components of a Singleton Session bean.

Singleton Session Bean Specification 1-3

- ❑ A Singleton Session bean comprises a bean class and one or more optional business interfaces.

Bean class	It is a standard Java class which needs to be marked with <code>@Singleton</code> annotation.
Business interface	The business interface can be a local or remote. Singleton Session bean also supports no-interface local view for the clients deployed on the application server.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 11

Use slide 11 to explain the components of a Singleton Session bean. It comprises a bean class which has the instance variables and methods associated with the functionality of the Singleton bean.

The bean can be accessed through the business interface defined for it. It can have multiple business interfaces defined. Like Stateless Session beans, Singleton Session bean also supports no-interface view for accessing the bean.

Tips:

There are three important tasks which are to be handled while creating a Singleton Session bean.

The developer has to perform the following tasks while implementing a Singleton Session bean:

1. Initialize the Singleton Session bean
2. Handle concurrent access to the Singleton Session bean
3. Handle errors with respect to the bean

Slides 12 and 13

Let us understand the sample implementation of a Singleton Session bean.

Singleton Session Bean Specification 2-3



Following code snippet demonstrates the creation of Singleton Session bean:

```
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Singleton(name = "ItemCount")
public class ShoppingItemCount {
    private int counter = 0;

    // Increment number of shopper counter
    public void incrementCounter() {
        shopperCounter++;
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 12

Singleton Session Bean Specification 3-3



```
// Return number of shoppers
public int getCounter() {
    return Counter;
}
// Reset counter
public void initializeCounter() {
    shopperCounter = 0;
}
```

The code annotates the `ShoppingItemCount` class with `@Singleton` annotation. The `name` attribute of the annotation defines the `ejb-name` of the bean by which it is referenced by other beans in the container.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 13

Use slides 12 and 13 to demonstrate a sample implementation of a Singleton Session bean.

Tell them that the code annotates the `ShoppingItemCount` class with `@Singleton` annotation. The `name` attribute of the annotation defines the `ejb-name` of the bean by which it is referenced by other beans in the container.

The code shows three methods and an instance variable `counter` in Singleton Session bean as follows:

- `initializeCounter()` method initializes the value of the variable counter in the program. This initialization process can be done through a constructor also.
- `incrementCounter()` increments the value of the instance variable counter
- `getCounter()` method returns the value of the counter in the code.

Additional References:

For more example on Singleton Session bean, you refer the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/gipvi.html>

Slide 14

Let us understand the process of initializing the Singleton Session bean.

Initialize Singleton Session Bean

- By default, EJB container is responsible for initializing the Singleton Session bean instance.
- The bean developer can also configure when to initialize the Singleton Session bean instance.
- If the bean is annotated with `@Startup`, then the bean is initialized at application startup.
- This initialization is also known as **eager initialization**, where the container initializes the Singleton Session bean as soon as the application startup.
- Following code snippet demonstrates the usage of `@Startup` annotation:

```
@Startup
@Singleton
public class ShoppingItemCount {
    ...
}
```

Instructs the container to configure and initialize the Singleton Session bean during the startup process of the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 14

Use slide 14 to explain the process of initializing a Singleton Session bean.

The EJB container initializes the Singleton Session bean according to the application configuration. The initialization strategy to be used can be defined through deployment descriptors or annotations. The developer can choose either eager initialization or lazy initialization as the initialization strategy.

When a Singleton Session bean class is annotated with `@Startup` it implies that the bean should be instantiated at the application start up otherwise the bean might be instantiated at a later time in the application. When the bean is initialized at a later time in the application, then the initialization strategy is known as lazy initialization of the application.

Explain the code snippet demonstrating the usage of the `@Startup` method.

In-Class Question:

After you finish explaining the process of initialization of Singleton Session Bean, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



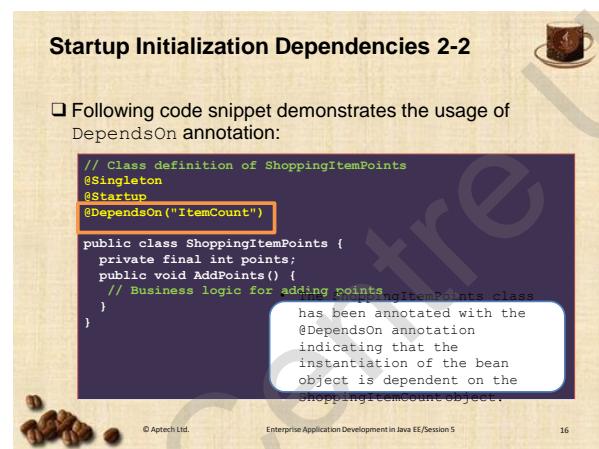
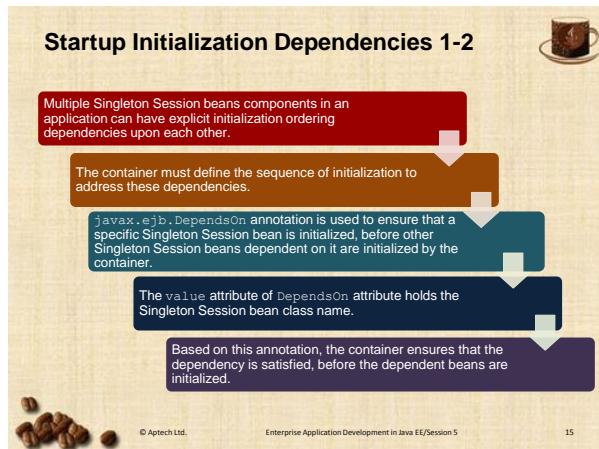
Which annotation indicates the actions to be performed by the container during application startup?

Answer:

`@javax.ejb.Startup`

Slides 15 and 16

Let us understand the initialization dependencies in Singleton Session beans.



Use slides 15 and 16 to explain initialization dependencies in the application.

An application can have multiple instances of Singleton Session beans. Initialization of these Singleton Session beans might be dependent on each other. Therefore, the order of initialization must be defined through the deployment descriptor.

`javax.ejb.DependsOn` annotation can be used to indicate the initialization dependencies of the application. The annotation ensures that all the required Singleton beans are initialized before initializing the current Singleton bean.

The `@DependsOn` annotation has a `value` attribute which defines the target and holds the Singleton Session bean class name. Based on this annotation, the container ensures that the dependency is satisfied, before the dependent beans are initialized. The `@DependsOn` annotation's `value` attribute is one or more strings that specify the name of the target singleton session bean. If more than one dependent singleton bean is specified in `@DependsOn`, the order in which they are listed is not necessarily the order in which the EJB container will initialize the target Singleton Session beans.

Use slide 16 to demonstrate the usage of `@DependsOn` annotation. Tell them that the code shows the `ShoppingItemPoints` bean which is dependent on the initialization of the bean `ItemCount`. Therefore, the container ensures that the bean `ItemCount` is initialized before the `ShoppingItemPoints` bean is initialized.

Slides 17 and 18

Let us understand the callback methods associated with Singleton Session beans.

LifeCycle Callback Methods 1-2



- ❑ Callback events handle the construction and destruction of a Singleton Session bean.
- ❑ Callback events are mapped to the following events:
 - PostConstruct**
 - It is denoted by `@PostConstruct` annotation and is fired after a bean instance is instantiated by the container, and before the invocation of the first business method defined in the class.
 - PreDestroy**
 - It is denoted by `@PreDestroy` annotation and is fired when the application is shutting down.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 17

LifeCycle Callback Methods 2-2



- ❑ Following code snippet demonstrates the PostConstruct and PreDestroy interceptor methods:

```

  ...
  @Singleton(name = "ItemCount")
  @Startup
  public class ShoppingItemCount {
  ...
  @PostConstruct
  public void applicationStartup() {
    System.out.println("ApplicationStartup - Initializing the
    counter variable to zero.");
    Counter = 0;
  }

  @PreDestroy
  public void applicationShutdown() {
    System.out.println("ApplicationShutdown Happening");
  }
}
  
```

The code handles the application startup and application shutdown in the callback interceptor methods.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 18

Use slides 17 and 18 to explain the lifecycle callback methods associated with the Singleton Session beans.

Using slide 17, explain the lifecycle of a Singleton Session bean. It is similar to that of Stateless Session bean. It cannot be passivated. Therefore, there are two types of callback methods associated with it – `PostConstruct` and `PreDestroy` methods. These methods are however invoked only once during the entire lifecycle of the application.

Slide 18 shows an example of the lifecycle callback methods in a Singleton Session bean. The `PostConstruct` method performs the initialization operations of the bean. It initializes the instance variable `Counter` in the method. Note, that the initialization strategy is eager initialization strategy in this case. The `applicationShutdown()` method is the `PreDestroy` callback method as implemented in the code.

Slides 19 and 20

Let us understand concurrency in case of Singleton Session bean.

Concurrency in Singleton Session Bean 1-2



- ❑ Every application has only one instance of a Singleton Session bean.
- ❑ When multiple clients access the same instance of the Singleton Session bean, then this results into **concurrency**.
- ❑ There are two ways by which the concurrent access to a singleton session bean can be controlled. These are as followed:
 - **Container-managed concurrent** - As the name suggests, the container is responsible for managing the concurrent access to the Singleton Session bean data or methods. This is the default concurrency management type.
 - **Bean-managed concurrent** – This container provides full access of the bean to the bean developer. Even the synchronization of bean state is managed by the bean developer.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 19

Concurrency in Singleton Session Bean 2-2



- ❑ The concurrency management method used for the current enterprise bean can be defined through the annotation `javax.ejbConcurrencyManagement`.
- ❑ The type of concurrency management can be specified through a `type` attribute whose value can be either set to `javax.ejbConcurrencyManagementType.CONTAINER` or `javax.ejbConcurrencyManagementType.BEAN`.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 20

Use slides 19 and 20 to explain concurrency management in a Singleton Session bean. Concurrency management is not required in other types of Session beans, as no other Session beans can be simultaneously accessed by multiple clients.

A singleton's client needs only a reference to a singleton, in order to invoke any business methods exposed by the singleton and doesn't need to worry about any other clients that may be simultaneously invoking business methods on the same singleton.

When creating a singleton session bean, concurrent access to the singleton's business methods can be controlled in two ways: container-managed concurrency and bean-managed concurrency.

Container-managed concurrency is the default strategy used by the applications. Bean-managed concurrency has to be defined by the application developer.

Using slide 20, explain the `ConcurrencyManagement` annotation. The concurrency management method used for the current enterprise bean can be defined through the annotation `javax.ejbConcurrencyManagement`.

The type of concurrency management can be specified through a `type` attribute whose value can be either set to `javax.ejbConcurrencyManagementType.CONAINER` or `javax.ejbConcurrencyManagementType.BEAN`.

In case, if no `@ConcurrencyManagement` annotation is present on the singleton implementation class, the EJB container default of container-managed concurrency is used.

Slides 21 to 23

Let us understand container-managed concurrency.

Container Managed Concurrency 1-3



Associates each business method of the Singleton Session bean with a lock.
 Annotations are used to define the type of the lock to be acquired.
 `javax.ejb.Lock` is used to specify the required locks.

- It has an attribute `javax.ejb.LockType`, which accepts the lock type as READ or WRITE for the bean methods.
- READ lock provides shared access and WRITE lock provides exclusive access.
- For example, marking the business method with `@Lock(LockType.READ)` grants shared access to the bean method.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 21

Container Managed Concurrency 2-3



Following code snippet demonstrates container-managed concurrency applied to the Singleton Session bean:

```

...
@Singleton(name = "WebsiteVisitCount")
@Startup
@ConcurrencyManagement(ConcurrencyManagementType.CONAINER)

public class WebsitevisitCount{
    ...
    private int Counter;

    // Increment number of visitors
    @Lock(LockType.WRITE)
    public void incrementCounter() {
        Counter++;
    }
}
```

The `@ConcurrencyManagement` annotation specifies that the concurrency type is set to `CONTAINER`.

- The `@Lock(LockType.WRITE)` acquires the WRITE lock which blocks the access of the method for all other clients, while one client is accessing it.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 22

```
// Return number of visitors
@Lock(LockType.READ)
public int getVisitorCount() {
    return Counter;
    . . .
}
```

The @Lock(LockType.READ) annotation is specified at the method-level to the getVisitorCount() which returns the number of users visited to the site.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 23

Use slides 21 to 23 to explain container-managed concurrency of a Singleton Session bean.

If a singleton uses container-managed concurrency, the EJB container controls client access to the business methods of the singleton. The container manages the concurrent access to the Singleton Session bean by associating the business methods with a lock. The methods can be defined with either a shared read lock or exclusive write lock.

There are two types of locks – READ and WRITE. When a client tries to read the data and does not intend to modify any of the instance variables of the Singleton Session bean then, the client acquires a READ lock. READ lock is also known as a shared lock as multiple clients can simultaneously acquire this lock and access the instance variables of Singleton Session bean.

WRITE lock is acquired by the client when it intends to modify the instance variables. When a client acquires a WRITE lock on instance variables, then no other client can access those instance variables. These type of locks are also known as exclusive locks. If a method is of locking type WRITE, client access to all the singleton's methods is blocked until the current client finishes its method call or an access timeout occurs. When an access timeout occurs, the EJB container throws a javax.ejb.ConcurrentAccessTimeoutException.

Java EE provides implementation of both types of locks. Developers can use the implementation of these locks in their application through annotations such as javax.ejb.Lock.

The type of the lock can be defined through the attribute javax.ejb.LockType.

Use slides 22 and 23 to explain the usage of different types of locks with methods.

The code shown on the slides shows two methods incrementCounter() and getCounter(). The method incrementCounter() intends to modify the variable counter, that is, it has to perform a WRITE operation on the instance variable counter. In order to perform this operation, it has to exclusively access the variable hence it has to acquire a WRITE lock.

The method getVisitorCount() is reading the value of the instance variable counter. It has to therefore, acquire a READ lock on the instance variable counter. This lock can be shared with other clients of the application.

Slide 24

Let us understand bean-managed concurrency.

Bean Managed Concurrency

- ❑ Bean managed concurrency is specified through the annotation `ConcurrentManagementType.BEAN`.
- ❑ It is the responsibility of the bean developer to synchronize the state of the Singleton Session bean to avoid synchronization errors occurring due to the concurrent access.
- ❑ Bean developer uses `synchronized` and `volatile` primitive types for synchronization.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 24

Use slide 24 to explain bean-managed concurrency. In this case, the developer has to explicitly define the concurrency management techniques to be implemented while accessing Singleton Session bean.

The developer of the singleton is responsible for ensuring that the state of the singleton is synchronized across all clients.

Developers use `synchronized` and `volatile` blocks of code to implement concurrency control. When bean managed concurrency control mechanism is used, the developer is responsible for incorporating the concurrent access mechanism in the bean class. It is the responsibility of the bean developer to synchronize the state of the Singleton Session bean to avoid synchronization errors occurring due to the concurrent access.

Tips:

If a Singleton Session bean encounters an error when initialized by the EJB container, that Singleton instance will be destroyed. Unlike other enterprise beans, once a singleton session bean instance is initialized, it is not destroyed if the singleton's business or lifecycle methods cause system exceptions. This ensures that the same singleton instance is used throughout the application lifecycle.

In-Class Question:

After you finish explaining the bean-managed concurrency, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 In bean-managed concurrency, who is responsible for ensuring that the state of the singleton is synchronized across all clients?

Answer:

Developer

Slides 25 and 26

Let us understand concurrent access timeout.

Concurrent Access Timeout 1-2



When a client has an exclusive lock, other clients are blocked from accessing the bean.

A client cannot hold the lock for unlimited time.

`@AccessTimeOut` annotation can be used to specify the maximum amount of time for which a client can be blocked.

`AccessTimeOut` annotation has two attributes `value` and `timeUnit`.

- The default time unit specified in the `value` attribute is in milliseconds.
- The developer can change it to one of the constants provided in the `java.util.concurrent.TimeUnit`.
- If the `AccessTimeOut` value specified is -1, then it indicates that the client will block indefinitely until it gains access to the bean.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 25

Concurrent Access Timeout 2-2



Following code snippet demonstrates the configuration of access time value for the Singleton Session bean:

```
// Increment number of visitors
@Lock(LockType.WRITE)
@AccessTimeout(value=120000)
public void incrementCounter() {
    Counter++;
}
```

- The code specifies the access time out for the `incrementCounter()` method to 120000 milliseconds.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 26

Use slides 25 and 26 to explain the concept of concurrent access timeout.

When a client is holding an exclusive lock on an instance variable, other clients which are trying to gain access to the variable have to wait until the client which is currently holding the lock releases it. However, the clients cannot wait indefinitely as it will block other processes in the application. Therefore, it is essential to set an upper limit on the time period for which a client will wait to acquire the lock. This upper limit on the waiting time is concurrent access timeout.

Developers can set the access timeout value through `@AccessTimeOut` annotation.

There are two attributes associated with the `AccessTimeOut` annotation; they are `value` and `timeUnit`. The `timeUnit` attribute is optional. The default time unit specified in the `value` attribute is in milliseconds. However, the developer can change it to one of the constants provided in the `java.util.concurrent.TimeUnit`. The constants are `MILLISECONDS`, `MICROSECONDS`, or `SECONDS` which can be specified in the `timeUnit` attribute.

The code shown on slide 26 shows an access timeout value of 12000 milliseconds to acquire an exclusive lock on the counter variable before incrementing the **counter**.

In-Class Question:

After you finish explaining about access timeout, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



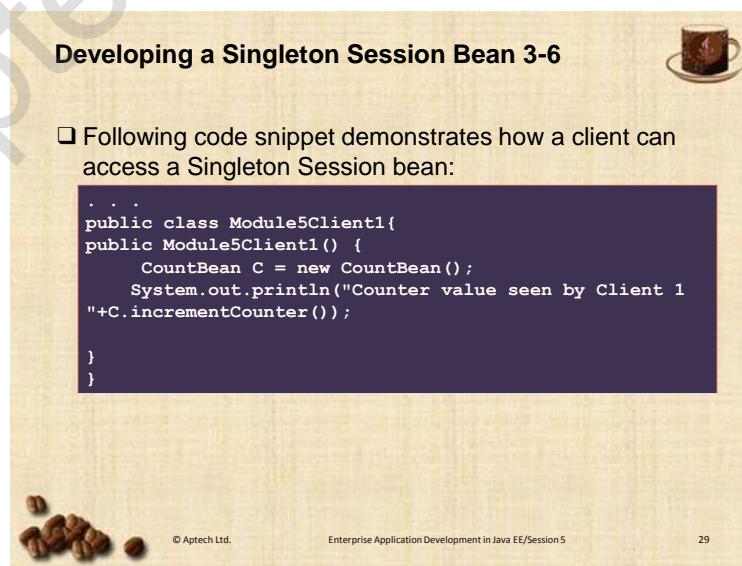
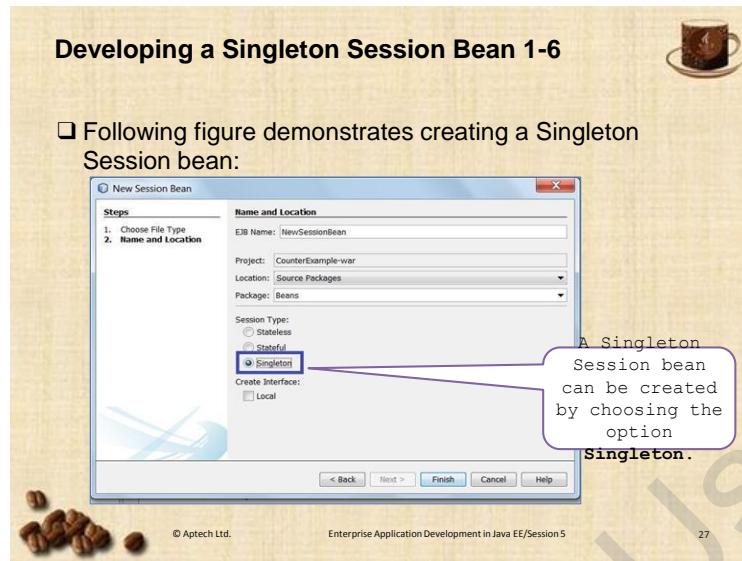
Why should developers set an access timeout?

Answer:

It is essential to set an upper limit on the time period for which a client will wait to acquire the lock. This upper limit on the waiting time is concurrent access timeout. Setting an access timeout value will improve the application performance.

Slides 27 to 32

Let us understand the process of creating a Singleton Session bean through NetBeans IDE.



Developing a Singleton Session Bean 4-6



- Following code demonstrates how a second client accesses the Singleton Session bean:

```
...
public class Module5Client2{

    public Module5Client2() {
        CountBean C = new CountBean();
        System.out.println("Counter value seen by
Client 2 "+C.incrementCounter());
    }
}
```

- Earlier code snippet and the given code snippet are two Java classes that invoke the Singleton Session bean in the constructor of the class.
- Whenever an object of the client class is invoked, the Singleton Session bean is accessed.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 5

30

Developing a Singleton Session Bean 5-6



- Following code snippet demonstrates multiple clients accessing a Singleton Session bean:

```
package Beans;
public class ClientRequest {
    public static void main(String[] args) {
        Module5Client1 m1 = new Module5Client1();
        Module5Client2 m3 = new Module5Client2();
        Module5Client1 m2 = new Module5Client1();
        Module5Client2 m4 = new Module5Client2();

    }
}
```

- The code creates four instances of clients which are equivalent to client requests to access the Singleton Session bean.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 5

31

Developing a Singleton Session Bean 6-6



- Following figure shows the execution of multiple client requests by the Singleton Session bean:

```
Output
Java DB Database Process >> GlassFish Server >> CounterExample
run:
Counter value seen by Client 1 11
Counter value seen by Client 2 12
Counter value seen by Client 1 13
Counter value seen by Client 2 14
BUILD SUCCESSFUL (total time: 1 second)
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 5

32

Use slides 27 to 32 to explain the process of creating a Singleton Session bean.

The process of creating a Singleton Session bean is similar to that of other Session beans. The NetBeans IDE wizard allows the developer to choose the type of Session bean being created by selecting appropriate option in the wizard as shown in the figure given on slide 27.

Use slide 28 to demonstrate the code of a Singleton Session bean. According to the code, the bean is accessed only through local interface. The concurrency control of the bean is implemented through the container. The `incrementCounter()` method in the given code requires a WRITE lock to be acquired before executing.

Use slides 29 and 30 to show the code for two different clients simultaneously trying to access the Singleton Session bean.

The two clients are simultaneously trying to execute the `incrementCounter()` method and acquire the WRITE lock on the corresponding instance variable.

Use slide 31 to show code which simulates the situation where multiple clients are trying to acquire a WRITE lock on the counter variable.

Use slide 32 to demonstrate the output of simultaneous execution. It will be more meaningful if you execute this code in NetBeans IDE during the class instead of demonstrating through the slides. Make relevant changes in the code to highlight the features of the Singleton Session bean.

Additional References:

To create Singleton Session bean in NetBeans IDE, you can execute one more application, as shown in the following link: <http://mrbool.com/how-to-create-singleton-session-bean-in-java/29029>

Slide 33

Let us summarize the session.

Developing a Singleton Session Bean 6-6

```

Output
Java DB Database Process | GlassFish Server | CounterExample
run:
Counter value seen by Client 1 11
Counter value seen by Client 2 12
Counter value seen by Client 1 13
Counter value seen by Client 2 14
BUILD SUCCESSFUL (total time: 1 second)

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 5 32

Using slide 33, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

5.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concepts of messaging through Java Messaging Services (JMS) and Message-driven bean that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 6 – Introduction to Messaging

6.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

6.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the messaging concept and its architecture
- Describe Java Messaging Service API
- Describe various messaging models supported by JMS
- Explain the working of Message-driven beans
- Describe how to create and configure a Message-driven bean in Java EE application

6.1.2 Teaching Skills

To teach this session successfully, you should be aware of messaging systems which are used in enterprise applications and their implementation using JMS API. You should have good understanding about the architecture of messaging framework provided by Java EE. You should be aware of the different messaging models supported by JMS API.

The session also covers the working of the Message-driven beans and their implementation in enterprise applications. You should be aware of how a developer can implement different messaging models in the Java enterprise applications and finally, how to configure the Message-driven beans to asynchronously process the messages on the application server.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❑ Describe the messaging concept and its architecture
- ❑ Describe Java Messaging Service API
- ❑ Describe various messaging models supported by JMS
- ❑ Explain the working of Message-driven beans
- ❑ Describe how to create and configure a Message-driven bean in Java EE application

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 2

Tell them that they will be introduced to the need for message-oriented architecture and then, understand how to use Java Messaging Service (JMS) API for messaging applications. Tell them they learn about different messaging models provided by the JMS API and the implementations.

Finally, they will implement a simple messaging application using both the messaging models. The session also explains how to develop Message-driven bean which can process messages asynchronously in the Java enterprise applications.

6.2 In-Class Explanations

Slide 3

Let us understand the purpose of a messaging system.

Introduction

- ❑ Java application architecture is loosely coupled.
- ❑ Components are independent of each other.
- ❑ **How the components communicate?**
 - Technologies such as:
 - Remote Method Invocation (RMI)
 - Common Object Request Broker Architecture (CORBA)
 - Component Object Model (COM) or Distributed Component Object Model (DCOM)
 - Allow communication between distributed components or objects.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 3

Use slide 3 to explain the purpose of a messaging system.

Java enterprise application is developed on a component-based framework. The application is designed as an aggregation of independent components. Each component is defined to perform a specific function. The components communicate with each other and invoke each other to perform a function in the application. This architecture is also termed as loosely coupled because each of the components is relatively independent of each other. The main advantage of loosely coupled components is that they need not be aware about each other's type, location, or implementation.

The application components, therefore, need a mechanism to communicate with each other and function together. Earlier versions of Java used mechanisms such as Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA) which would enable communication between the distributed applications components developed in Java.

Discuss with them about distributed objects they must be have heard or known on Windows platform provided by Microsoft. Then, introduce to them with Component Object Model (COM) or Distributed Component Object Model (DCOM) technologies that facilitate communication of components on Windows platform.

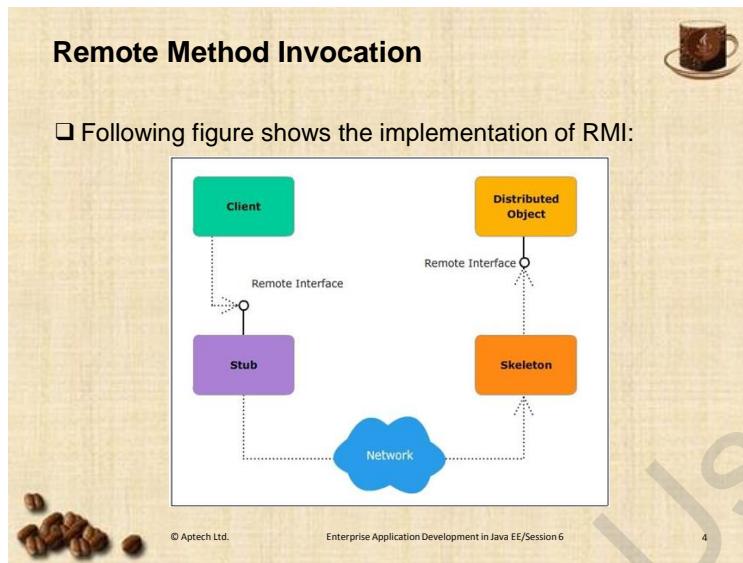
Additional References:

To know more about COM technology, visit to the following link:

<https://www.microsoft.com/com/default.mspx>.

Slide 4

Let us understand remote method invocation.



Use slide 4 to explain Remote Method Invocation (RMI).

The RMI mechanism provides communication between the Java objects. It allows object running in one Java virtual machine to invoke methods on an object running in different JVM machine.

Both client and server maintain a proxy for remote communication. This proxy on the client-side is known as Stub and the proxy on the server-side is known as Skeleton. These proxies do not block the client or server processes, while they are waiting for response from a remote object.

When a client invokes a method on the server, the request from the client is sent to the Stub. After sending the request to the Stub, the client process carries out other processes. The Stub is responsible for sending the request to the server and retrieving the response. If there are any intermediate network failures, Stub handles it by resending the request or taking other actions. Client process is unaware of these network failures.

Similarly, the Skeleton on the server-side is responsible for receiving client requests and forwarding them to the server. The Java object residing on the server processes the request and hands over the response to the Skeleton. The Skeleton then sends the method execution result to the client. On the client-side the Stub receives the response message.

Slide 5

Let us understand distributed communication in enterprise applications.

Distributed Communication in Enterprise Applications 1-2

- ❑ Sun Microsystems introduced RMI mechanism which provides a native way to communicate distributed objects running in the different JVMs.
- ❑ RMI is an extension of Java Remote Invocation over Internet Inter-ORB Protocol (RMI-IIOP).
 - The use of RMI-IIOP resulted in the interoperability between the applications based on CORBA architecture.
 - Later on, RMI-IIOP became the protocol for EJBs to communicate and hence, served as a foundation for EJB.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 S

Use slide 5 to explain the process of distributed communication in applications.

There is an extension to Java RMI which enables communication of application components written in different programming languages. This extension is RMI-IIOP where Internet Inter-ORB Protocol (IIOP) is a protocol which enables for distributed programs written in different programming languages to communicate over the Internet. This architecture is based on client-server architecture. The RMI-IIOP was written by Object Management Group (OMG) to provide a standard way of communication between the CORBA products provided from different vendors.

RMI-IIOP serves as the basis for EJB components to communicate with each other. It provides a standard way of communication between the CORBA products provided from different vendors. The use of RMI-IIOP resulted in the interoperability between the applications based on CORBA architecture. Later, when the developers of EJB felt a similar need of achieving interoperability among the EJB containers provided by different vendors. They adopted RMI-IIOP over RMI for providing communication between the different EJB components. RMI-IIOP became the protocol for EJBs to communicate and hence, served as a foundation for EJB.

Slide 6

Let us understand the reason for RMI-IIOP that makes it unsuitable for EJB.

Distributed Communication in Enterprise Applications 2-2

- ❑ Following are the features of RMI-IIOP which make it unsuitable for usage with EJB components:
 - Cannot support asynchronous communication.
 - Client and server are not decoupled.
 - Does not provide service reliability.
 - Cannot support communication between a client and multiple servers.

Since RMI-IIOP lacks the mentioned features, messaging was introduced for EJB applications.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 6

Use slide 6 to explain the features required by EJB applications which are not present in RMI-IIOP.

Discuss with them the limitations of RMI-IIOP that were addressed, later on, by Messaging.

Some of the limitations of RMI-IIOP that are as follows:

- **Asynchronous Communication** – RMI-IIOP does not support asynchronous communication. This means the RMI-IIOP client has to wait for the server response, till the server completes its work, and sends the result back to the client. Hence, client keeps on waiting, before continuing its processing.
- **Decoupling** – The RMI-IIOP client needs to be aware with the server for accessing the remote objects through references. This results in the dependency of client on the server. In case if the server has to be removed, then this will affect the client directly.
- **Reliability** – When the RMI-IIOP client invokes the component on the server, the server must be running. In case if the server crashes, then the data might be lost and client may not complete its operations.
- **Support for Multiple Producer and Consumer** – The RMI-IIOP restricts the communication between a single client and a single server at any given of time. It lacks the broadcasting of events from multiple clients to multiple servers.

Slide 7

Let us understand the concept of messaging in the enterprise applications.

Messaging 1-2

- ❑ Messaging is based on the concept of Message Oriented Middleware (MOM) which serves as a middleman placed between the client and the server.
- ❑ Messaging implements asynchronous message processing.
- ❑ Following figure compares the implementation of RMI against Messaging:

```

graph LR
    subgraph RMI [RMI]
        A1[Application] -- "Remote Method Invocation" --> A2[Application]
    end
    subgraph Messaging [Messaging]
        A3[Application] --> MM[Message Middleware]
        MM --> A4[Application]
    end

```

The diagram illustrates two communication models. The top section, labeled 'RMI', shows two green rectangular boxes labeled 'Application' connected by a horizontal arrow labeled 'Remote Method Invocation'. The bottom section, labeled 'Messaging', shows a similar setup but with a red rectangular box labeled 'Message Middleware' positioned between the two applications, indicating that messages pass through this middleware component.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 7

Use slide 7 to explain the concept of messaging in the enterprise applications.

Earlier applications were based on client-server architecture, but EJB applications are based on n-tier application architecture or three-tier architecture. The middle tier of the application has the business logic of the application. Therefore, most of the business components of the application reside in the middle tier.

Messaging is required for establishing communication between the components in the middle tier. The communication among the components is expected to be asynchronous; therefore, the messaging architecture introduced a Message-Oriented Middleware (MOM). It is responsible for receiving messages from a sender component and delivers the message to a receiver component.

The figure given on slide 7 shows the location of the middleware among the application components.

Tips:

Messaging is an alternative to RMI. Messaging middleman sits between the client and server. The middleman receives messages from one or more message producers and broadcasts them to one or more message consumers. The producer can send a message and then, continue processing. This mechanism is called asynchronous processing.

Slide 8

Let us understand the features of messaging over RMI-IIOP.

Messaging 2-2

- ❑ Messaging implements the following features to overcome the short comings of RMI-IIOP:
 - Asynchronous processing.
 - Decoupling of message senders and consumers.
 - Reliable message delivery system based on MOM.
 - Support for multiple message producers and consumers.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 8

Use slide 8 to explain the features provided by MOM over RMI-IIOP in the enterprise applications.

- **Non-blocking request processing or Asynchronous processing:** While executing a request, the messaging client need not block other tasks that it was processing.
- **Decoupling:** In Message Oriented Middleware (MOM) system, the message sender is not required to know the message receivers as it addresses the messaging system. Message senders are decoupled from message consumers and are not affected by change in customers.
- **Reliability:** In MOM system, a message is delivered even if the customer is not available for a specific period of time. The message is sent to the MOM system which routes the message to the consumer when active. In RMI-IIOP, an exception is thrown when the server is down.
- **Support for Multiple Producer and Consumer:** MOM system can accept messages from multiple message producers and broadcast it to multiple consumers.

In-Class Question:

After you finish explaining features provided by MOM, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Explain the decoupling feature of MOM-based architecture.

Answer:

In MOM system, the message sender is not required to know the message receivers as it addresses the messaging system. Message senders are decoupled from message consumers and are not affected by change in customers.

Slide 9

Let us understand the functions of MOM.

Message Oriented Middleware (MOM) 1-4

- ❑ Refers to an infrastructure that supports messaging.
- ❑ Can be defined as a software that enables asynchronous message exchange between system components.
- ❑ The software stores the message in the location specified by the sender and acknowledges immediately.
- ❑ The message sender is known as the **producer**.
- ❑ The location where the message is stored is known as the **destination**.
- ❑ The software components can retrieve the stored messages and are known as message **consumers**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 9

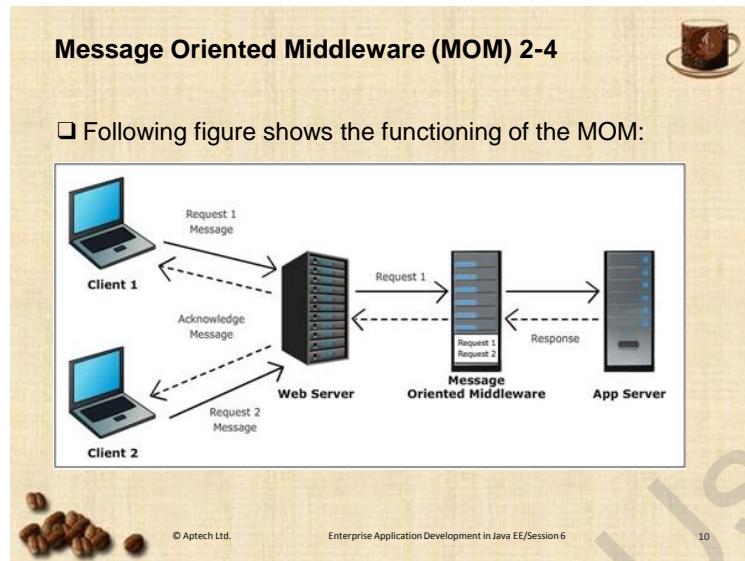
Use slide 9 to explain the features of MOM.

Message Oriented Middleware (MOM) provides the required infrastructure to provide the messaging function within an enterprise application. It supports the function of sending messages and receiving messages in distributed applications. In the context of MOM, message sender components are referred as message producers and message receiving components are referred as message consumers.

MOM based messaging system has an intermediate messaging provider which receives messages and forwards those messages to appropriate destinations, enabling asynchronous exchange of messages. The messages are retained by the messaging provider, until the destination is not available.

Slide 10

Let us understand the functioning of MOM.



Use slide 10 to explain the process of message exchange among the components in a MOM-based system.

As shown in the figure given on slide 10, Client 1 and Client 2 represents two communication components that intend to communicate in the application. The Web server indicates that the Web application communicates with the MOM infrastructure to enable asynchronous communication.

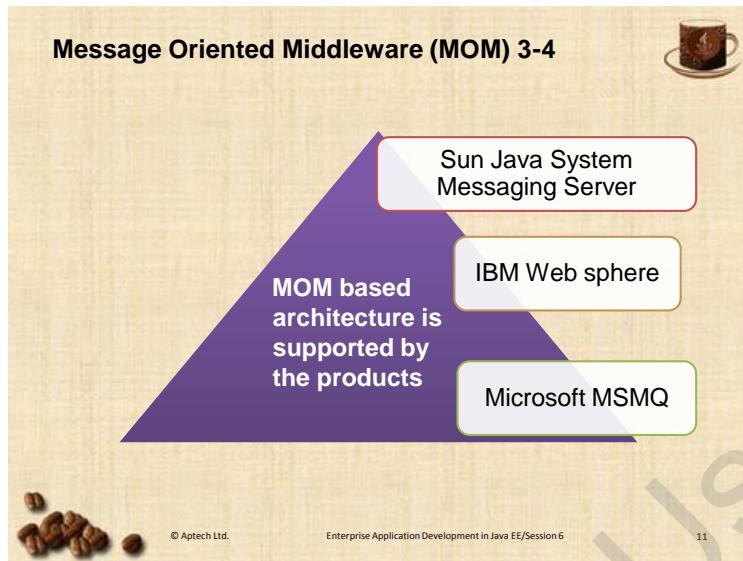
Note: The message destination should hence be identified through an IP address.

When the message is received by the Web server it forwards it to the MOM infrastructure. If the message requires some processing then the MOM component processes it by interacting with the application server. The message is then either stored on the MOM server till Client 1 is available or is forwarded to Client 2 if it is available.

Client 2 may send a response to the message received from Client 1.

Slide 11

Let us look at various messaging infrastructure provided by different application server vendors.



Use slide 11 to introduce various messaging infrastructure provided by different application server vendors.

Sun Java System Messaging Server is now Oracle messaging server. It is part of Java Enterprise System and capable of supporting large enterprises with mail users. It can be used across different operating systems.

IBM WebSphere refers to a set of enterprise products which is a mix of software and middleware products. It also has a mailing system incorporated in it.

MSMQ stands for Microsoft Message Queue, it is compatible with Windows operating system.

Tips:

The messaging infrastructure provided value-added services such as guaranteed message delivery, fault tolerance, load balancing, managing subscribers, and so on.

Slide 12

Let us understand the disadvantages of MOM.

Message Oriented Middleware (MOM) 4-4

- ❑ MOM implementation has the following disadvantages:
 - Any server supporting MOM-based middleware has its own APIs which is specific to a vendor.
 - MOM APIs are not portable to other messaging systems.
 - Developers have to learn the proprietary messaging API to incorporate messaging functionality.

The Java EE platform provides a vendor-neutral API known as Java Messaging Service (JMS).

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 12

Use slide 12 to highlight the disadvantages of MOM.

Tell them that every product providing MOM support on its application servers has its own API to work with the messaging infrastructure. This created vendor dependency to use messaging system, as code is not portable across the MOM infrastructure. Thus, Sun microsystems designed a standard messaging API which will eliminate the disadvantages of MOM-based products faced over the years for proprietary APIs. This Messaging API is Java-based messaging API which is platform independent and can provide communication between the components in the enterprise applications.

Disadvantages of MOM Architecture

- Any server supporting MOM-based middleware has its own APIs which is specific to a vendor.
- MOM APIs are not portable to other messaging systems.
- Developers have to learn the proprietary messaging API to incorporate messaging functionality in applications. Developers who have knowledge of Java messaging system can implement messaging component of the application and make it compatible with component deployed on any platform.

Slides 13 and 14

Let us understand Java Messaging Service (JMS).

Java Messaging Service (JMS) 1-2




© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 13

- ❑ JMS supports messaging among the application components on the Java EE platform.
- ❑ JMS eliminates the need to learn the vendor-specific MOM-based APIs to perform communication between the components in the enterprise applications.
- ❑ JMS allows asynchronous communication between the components through messages.

Java Messaging Service (JMS) 2-2




© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 14

JMS API is divided into two parts:

- JMS API**
 - Provides the functionality of creating, sending, receiving, and reading messages among the application components through a set of interfaces.
- Service Provider Interface (SPI)**
 - Used as a plug-in for the MOM implementation on the server.

Use slides 13 and 14 to introduce JMS. JMS is an API which is a part of Java EE.

JMS enables developers to implement the messaging system as a part of an enterprise application or as an independent application. Since, it provides asynchronous communication it can be used in applications where the components are loosely coupled.

There are two components of Java Messaging Service – JMS API and Service Provider interface. The JMS API is a set of classes and interfaces which are used by the developer to implement the functions of the client. API classes are used to perform operations such as sending a message, sending an acknowledgement for a message, and so on.

Service Provider interface is the MOM implementation on the server. This can be present on the application server or on a separate messaging server. It provides all the required structures to store the messages, implements the mail boxes on the server and is responsible for all the server related aspects of the application.

Slide 15

Let us understand the communication models supported by JMS.

JMS Communication Models

Based on the pattern of communication there are two messaging models supported by JMS:

- Publish-Subscribe model
- Point-to-Point messaging model

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 15

Use slide 15 to introduce the communication models supported by JMS.

JMS supports two different communication models – Publish-Subscribe model and Point-to-Point messaging model.

Publish-Subscribe communication model has one producer and multiple consumers for a message. This is used when the application has to broadcast messages to multiple message consumers.

Point-to-Point communication model is used when there is one producer and one consumer for a message.

In-Class Question:

After you finish explaining JMS communication models, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which model can be used when the message has to be published to multiple consumers?

Answer:

Publish-Subscribe communication model

Slides 16 and 17

Let us understand Publish-subscribe model.

Publish-Subscribe Model 1-4



- Multiple message producers can communicate with multiple message consumers.
- Messages are sent through a virtual channel called **Topic**.

Working of Publish-Subscribe Model

- Topic** is a logical destination object which contains messages from different sender components.
- The components which is sending the message to the topic is said to be a **Publisher**.
- In order to access messages from the topic, the components should subscribe to the topic.
- Subscriber** is the term used to refer to a message consumer component which has subscribed to receive all the messages from topic destination.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 16

Publish-Subscribe Model 2-4



- Following figure shows how different components communicate in a publish-subscribe model:

The MOM system maintains the list of subscribers for each Topic.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 17

Use slides 16 and 17 to explain Publish-Subscribe communication model.

Use slide 16 to explain the Publish-Subscribe model used in applications when there is one message producer and multiple message consumers. Message producer in this case is known as Publisher and message consumer is known as Subscriber. The message server has a topic which is a logical identity for a set of messages. Each message published by a publisher is posted onto a topic on the messaging server. The subscribers who are message consumers access the message from the topic on the server.

Topic is a logical destination object which contains messages from different sender components.

The topic sends a copy of the message received to all the subscribers. Subscriber is the term used to refer to a message consumer component which has subscribed to receive all the messages from topic destination. The MOM system is responsible to maintain the list of subscribers on the server.

Use slide 17 to explain the components of Publisher-Subscriber model.

Explain the figure given shown on slide 17 to understand the components of Publisher-Subscriber model.

Topic is the logical message destination on message server. A message server can have multiple topics hosted on it. Message publisher can post a message onto the topic. Each topic may have more than one publisher.

Message subscriber reads the message from the topic. Each topic can have multiple message subscribers.

In-Class Question:

After you finish explaining JMS communication models, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which messaging destination is used for Publisher-Subscriber model?

Answer:

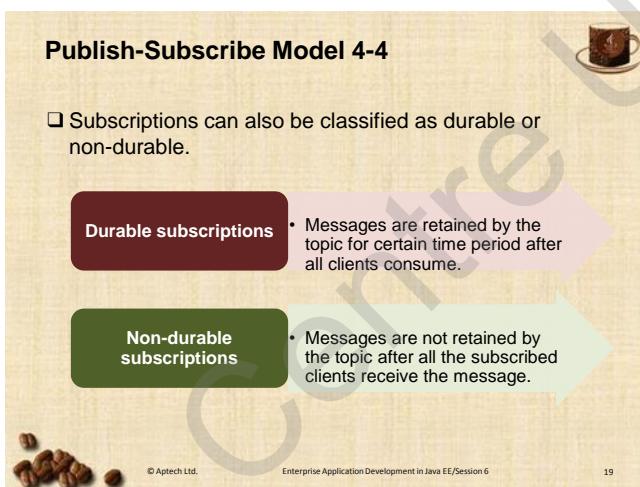
Topic

Additional References:

For more information on JMS communication models, visit the following link:
http://www.onjava.com/pub/a/onjava/excerpt/ejb3_ch13/?page=4.

Slides 18 and 19

Let us understand the variants of message subscriptions.



Use slides 18 and 19 to explain different types of subscriptions in Publish-Subscribe communication model.

There are two types of subscriptions defined in Publish-Subscribe communication model - Shareable subscriptions and Non-Shareable subscriptions.

A sharable subscription implies that the message can be received by a set of consumers who have agreed to share the subscription, whereas non-sharable subscription implies that only one subscriber can receive a message through subscription.

Use slide 19 to explain durable and non-durable subscriptions.

The messages sent to a topic cannot be retained forever on the messaging server. There should be a strategy which determines when the message must be removed from the topic.

In case of durable subscriptions, the messages are retained for certain period of time after all the consumers with subscriptions have consumed the message. This time duration is according to the application configuration.

In case of non-durable subscriptions, the messages are retained in the topic until all the consumers with subscriptions consume the message and are deleted after that.

Slides 20 and 21

Let us understand point-to-point communication model.

Point-to-Point Model 1-2



❑ The communication is between a pair of components.
 ❑ The destination is called as a Queue.

Working of Point-to-Point Model

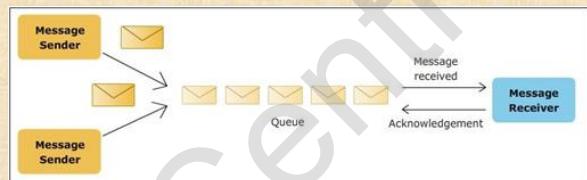
- ❑ There is only a single consumer for each message.
- ❑ A consumer can grab the message in the queue, but a given message is consumed only once by a consumer.
- ❑ Multiple producers can send a message to the queue.
- ❑ Only one consumer can consume a message from the queue.
- ❑ Messages are sent by the producers to a centralized queue and are distributed as First In First Out (FIFO).
- ❑ Receivers access the queue to retrieve the messages.
- ❑ Messages are retained in the queue until the consumer consumes it.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 20

Point-to-Point Model 2-2



❑ Following figure shows how different components communicate through point-to-point messaging model:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 21

Use slides 20 and 21 to explain Point-to-Point communication model in the application. Developers use Point-to-Point communication model when the communication has to take place between a single message producer and consumer.

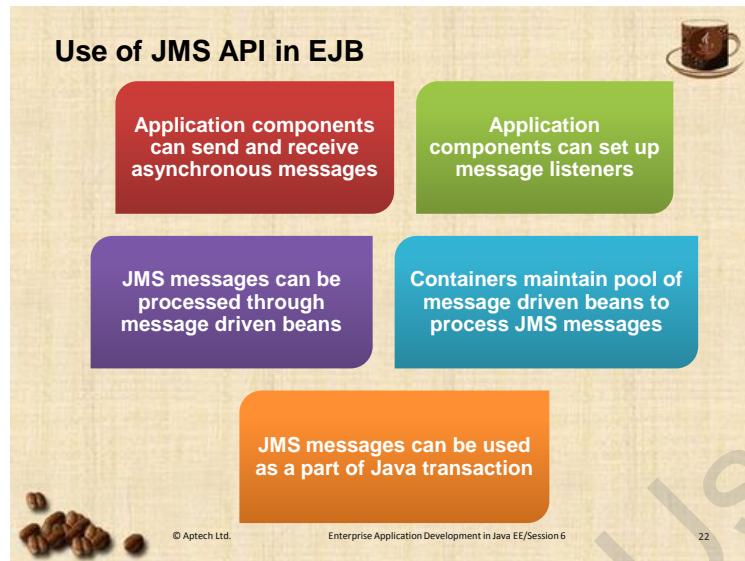
The messaging server maintains a Queue according to this communication model. Each message consumer has a Queue on the messaging server. When the message producer wants to send a message, the message is posted onto the corresponding queue of the consumer. Note that, messages are sent by the producers to a centralized queue and are distributed as First In First Out (FIFO).

The figure as shown on slide 21 shows multiple message producers posting messages onto the Queue of a message receiver. The message consumer receives the message from the queue and acknowledges the queue after the message has been successfully consumed. The message consumers can also configure a message listener to listen to the message queue and invoke appropriate methods when a message is received on the queue.

The message consumption is by default asynchronous in both the messaging model as the message consumers do not actively wait on the destination objects.

Slide 22

Let us understand the usage of JMS API in EJB applications.



Use slide 22 to explain usage of the JMS API in EJB applications.

- Various components such as application clients, enterprise beans, and Web components can send messages and asynchronously receive JMS messages.
- The API allows the application components to set up a message listener for processing the JMS messages. Message listeners are objects which respond to messages received and invoke appropriate components on receiving a message.
- Java EE also provides Message-driven beans which are enterprise components. These Message-driven beans are used to process JMS messages.
- Containers can maintain a pool of message-driven beans to enable concurrent processing of messages. This pool of message-driven beans facilitates different JMS messages simultaneously.
- JMS messages can also be part of a Java transaction.

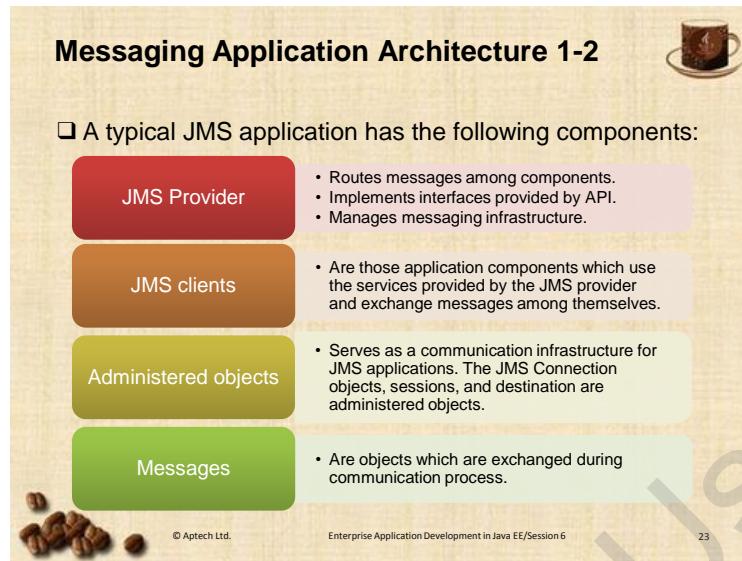
Additional References:

For more information on JMS model, visit the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html#bnceb>.

Slide 23

Let us understand messaging application architecture.



Use slide 23 to explain the messaging application architecture.

There are four components in a JMS application – JMS provider, JMS clients, Administered objects, and Messages.

JMS Provider provides the actual implementation of the messaging architecture. It provides the implementation of messaging infrastructure such as queues, topics, and so on. It also defines how messages will traverse from one component to another. There are various implementations of JMS providers.

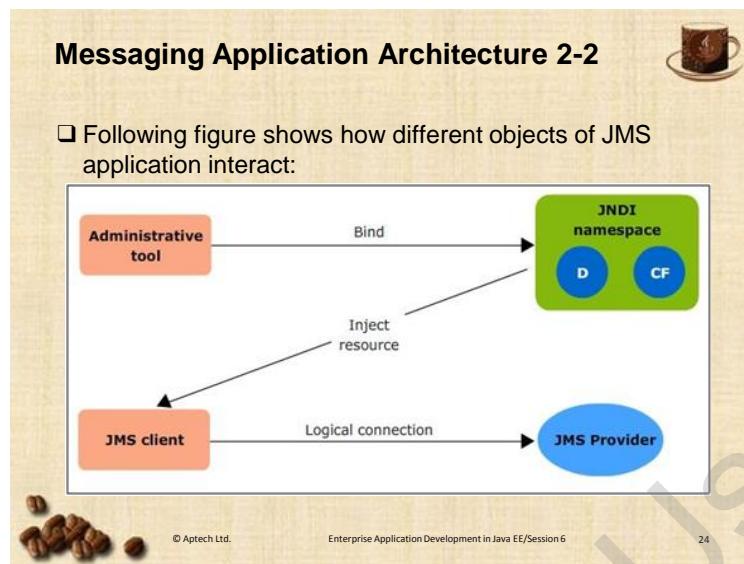
JMS clients are message producers and message consumers which use the services of the JMS Provider.

Administered objects refer to those entities which serve as communication infrastructure for the JMS application or JMS module of the application. JMS connection objects, sessions, and destinations are administered objects. These objects are instantiated according to the requirement of the application which happens when the application components want to communicate among themselves. They are logically injected into the application according to the requirement by the developers or administrators.

Messages are the objects which are exchanged among the components and supported by the messaging infrastructure.

Slide 24

Let us understand how various JMS components are associated with each other.



Use slide 24 to explain the interaction of various components of JMS.

Explain the figure as shown on slide 24 to explain the components of JMS. JMS client acquires Connection object and Destination object into the application. This resource injection is done through JNDI namespace. The JMS client obtains references of Destination and ConnectionFactory objects. The client accesses the service provided by the JMS Provider by establishing a connection with the JMS Provider through the administered objects.

Slide 25

Let us understand the JMS API Programming model.

JMS API Programming Model 1-8

Following are the objects which are part of a JMS application:

- Administered objects
- Connections
- Session
- JMSCContext
- JMS Message Producers
- JMS Message Consumers
- Messages

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 25

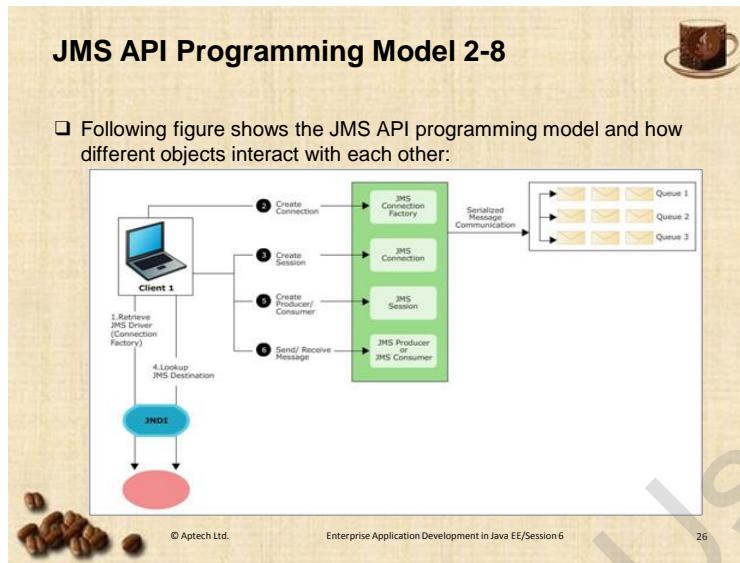
Use slide 25 to introduce the basic building blocks of a JMS application.

Destinations and ConnectionFactories used in JMS applications are better managed administratively than programmatically. These objects are accessed through portable interfaces. An administrator can configure administered objects in JNDI namespace. JMS clients can lookup these administered objects through JNDI API. ConnectionFactory is used to create Connections and Destination. In an object the client uses to specify the target of messages it produces and the source of the messages it would consume.

Connection is an object which encapsulates a virtual connection with a JMS provider. A Connection object can be a QueueConnection or a TopicConnection. A Session is a series of messages produced and consumed. JMSCContext defines the operational environment for a message exchange in the application. JMS Message Producers create messages and post them onto the destination. JMS Message consumers consume the messages from the destination. Messages are the objects which are exchanged among the Message Producers and Consumers.

Slide 26

Let us understand the functioning of JMS API programming model.



Use slide 26 to explain the working of JMS API programming model.

The communication starts with a JMS client, where the client acquires a Connection object and Destination object through JNDI namespace. After obtaining references for Connection and Destination objects, the client creates a session. The message producers and consumers are part of this session. They exchange series of messages throughout the session.

In-Class Question:

After you finish explaining the working of JMS API, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is exchanged among the Message Producers and Consumers?

Answer:

Message objects

Slide 27

Let us understand the components of the programming model.

JMS API Programming Model 3-8

JMS Connection Factories

- Are administered objects.
- Can be an instance of `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory`.
- Connection object when instantiated is a part of `JMSContext`.
- The Java EE server provides a JNDI name to access the `ConnectionFactory` object.

JMS Destination

- An administered object.
- Either a Queue or Topic.
- Can be created on the application server.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 27

Use slide 27 to explain JMS `ConnectionFactory` and JMS Destination.

`JMSConnectionFactories` are referred as administered objects according to the JMS API programming model. The developer can obtain a connection instance from the `ConnectionFactory`. This connection object further becomes a part of the `JMSContext`. A connection factory object can be an instance of `ConnectionFactory`, `QueueConnectionFactory`, or `TopicConnectionFactory` interface.

JMS Destination is also an administered object in a JMS application which is either a Queue or Topic. Administered objects can be created on the application server such as GlassFish through administrative tools provided by the server console or Admin Console provided by the NetBean IDE.

Slide 28

Let us understand the components of the JMS API programming model.

JMS API Programming Model 4-8

JMSContext objects

- Provide an application environment for JMS applications.
- Are created from a **ConnectionFactory** through **createContext()** method.
- Are used to create other objects of the applications.
- Are associated with a **Connection** and **Session** object.

JMS Message Producer objects

- Can be created using **createProducer()** method in the **JMSContext**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 28

Use slide 28 to explain JMSContext objects and JMS Message Producer objects.

JMSContext comprises a Connection object and a Session object. It provides an execution environment for message exchange. It is created from a **ConnectionFactory**.

JMSContext can further be associated with messages, message producers, and message consumers. It enables defining the message acknowledgement mode also for the application.

Message Producer objects are created within the JMSContext using **createProducer()** method.

Slide 29

Let us understand remaining components of the JMS API programming model.

JMS API Programming Model 5-8

JMS Message Consumers

- Created in a similar way as message producers along with the `JMSContext` instance.
- `setAutoStart()`, `receive()`, and `start()` methods can be used to alter the behavior of the consumer.

Messages

- Object of communication.
- Every message has a standard format.
- Each message is associated with a message header.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 29

Use slide 29 to explain JMS Message Consumers and Messages.

Message consumers are also created through `JMSContext`. The interface provides various methods to define the behavior of the message consumer.

A JMS message consumer can be created as follows:

...

```
JMSContext J = connectionFactory.createContext();
JMSConsumer JC = J.createConsumer(destination);
```

Once the destination is configured, the consumer receives the messages from the destination object. This behavior can be disabled by invoking the method `setAutoStart(false)`. After setting the auto start behavior to false to receive a message, `start()` method should be invoked by the consumer object and then `receive()` method must be explicitly called to receive the messages. JMS consumers can define message listeners on the destination objects.

Messages are objects of exchange among the application components. Each message has certain format and is associated with a message header.

Additional References:

To understand about the JMS API components, visit the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/bncest.html>

Slides 30 and 31

Let us understand JMS message format and fields in the message header.

JMS API Programming Model 6-8



- ❑ There are three parts of a message – message header, properties, and body.
- ❑ Message header holds all the control information of the message.
- ❑ Following table shows the interpretation of various fields in the message header:

Header field	Description
JMSDestination	This field defines the destination where the message is destined to. This is set by the <code>send()</code> method of message producer.
JMSDeliveryMode	This field defines the mode of message delivery, that is the method of persisting the message. This value is set by the <code>send()</code> method.
JMSDeliveryTime	This field of the message defines the time when the message sending process is initiated and also the delay time tolerable. This field is also set by the <code>send()</code> method.
JMSExpiration	This field defines the maximum time duration for which the current message is valid. This value is set by the <code>send()</code> method.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 30

JMS API Programming Model 7-8



Header field	Description
JMSPriority	This field defines the priority of the message and is set by the <code>send()</code> method of the message producer.
JMSMessageID	It is an unique identifier to identify the message and is set by the <code>send()</code> method.
JMSTimeStamp	This field specifies the time when the message was handed over to the JMS provider and is set by the <code>send()</code> method.
JMSCorrelationID	This field is set by the client and is used to link one message to the following message in a session.
JMSReplyTo	This field specifies the destination where the reply to the current message has to be sent.
JMSType	JMS messages can have data of different types. This field defines the data type of the message content and is set by the client.
JMSRedelivered	This field is set when the acknowledgement for the message is not received and as a result the message is redelivered.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 31

Use slides 30 and 31 to explain various fields in the message header and significance of each of the fields during message transmission.

Tell them that a JMS message header contains a number of predefined fields that contain values used by both clients and providers to identify and route messages.

Each header field has associated setter and getter methods, which are documented in the description of the Message interface. Some header fields are intended to be set by a client, but many are set automatically by the send or the publish method, which overrides any client-set values.

Then, explain the various header fields as shown on slide 31 that informs how the JMS Message Header field can be set.

Slide 32

Let us understand the different types of messages that can be exchanged.

JMS API Programming Model 8-8

- ❑ Message body implies actual content of the message.
- ❑ JMS supports following types of messages:
 - Text message
 - Map message
 - Bytes message
 - Stream message
 - Object message
 - Message

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 32

Use slide 32 to discuss the different types of messages that can be exchanged by different components of the enterprise application. These message types allow you to send and receive data in many different forms and which provide compatibility with existing messaging formats.

Then, explain the different message types supported by JMS.

Text Message – This comprises data as Java strings.

Map Message – The data has map objects, a map object is a key-value pair. The entities of a map object can be sequentially accessed through enumerator.

Bytes Message – A stream of un-interpreted bytes which match some data format such as JPEG and so on.

Stream Message – A stream of values which are of primitive types.

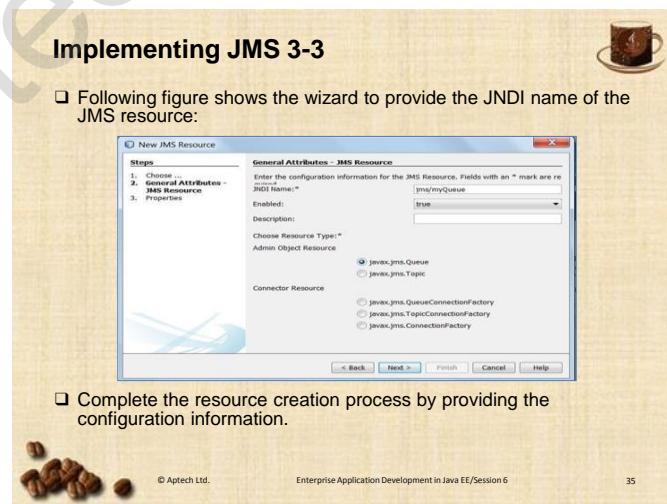
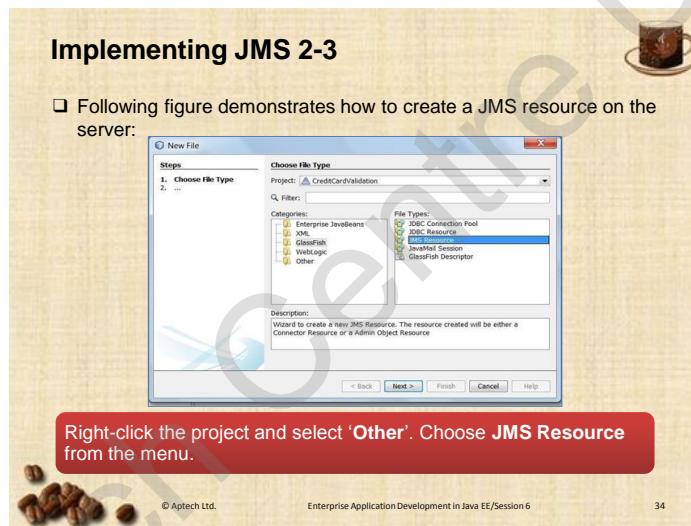
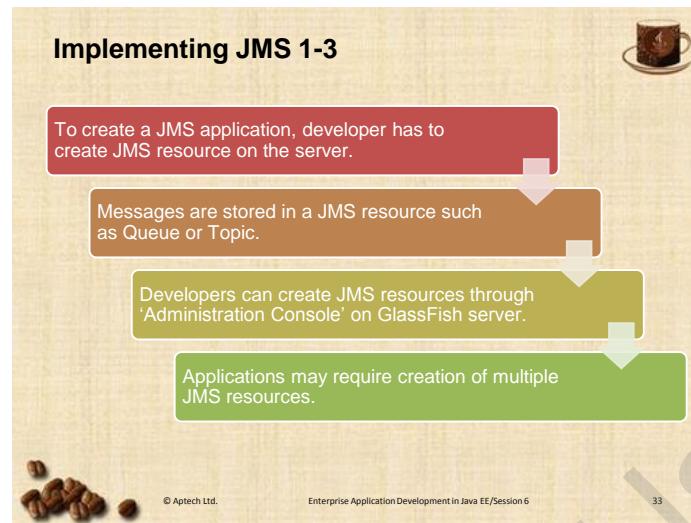
Object Message - A serializable object.

Message – Empty message body is also a valid message in JMS. This is used for administrative tasks.

JMS API has methods for creating and processing the messages of the mentioned message types. At the consumer end the message is received as a generic message object, which is later type casted onto appropriate data type.

Slides 33 to 35

Let us understand how to create JMS resources on a server.



Use slides 33 to 35 to explain the steps involved in creating a JMS resource in the application.

Use slide 33 to explain that JMS resource implies a JMS destination on the application server; here in this case it is GlassFish. The NetBeans IDE provides an '**Administration Console**' through which developers can add JMS resources to the application. The application may have more than one JMS resource.

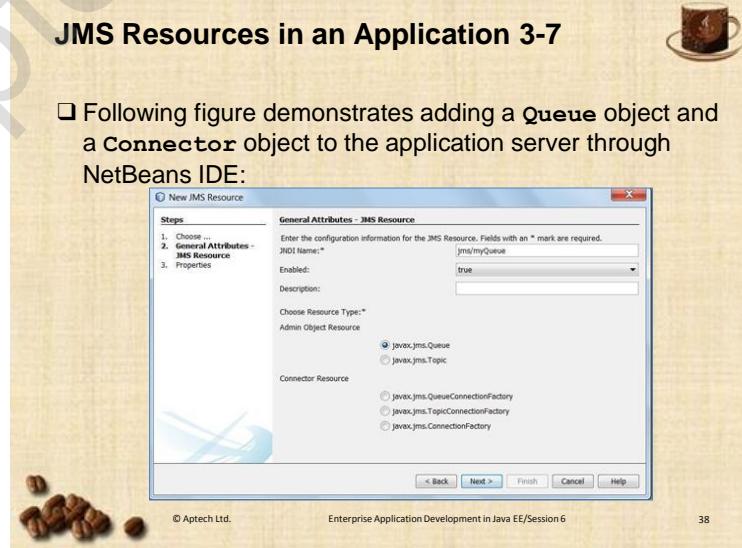
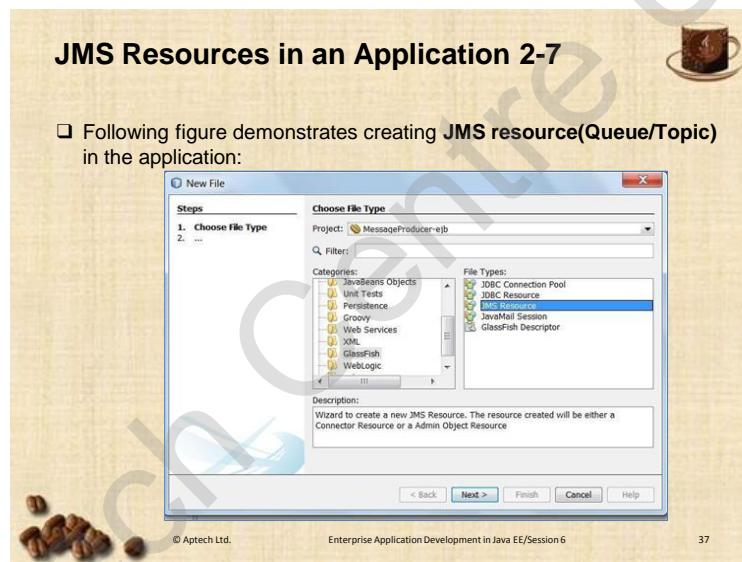
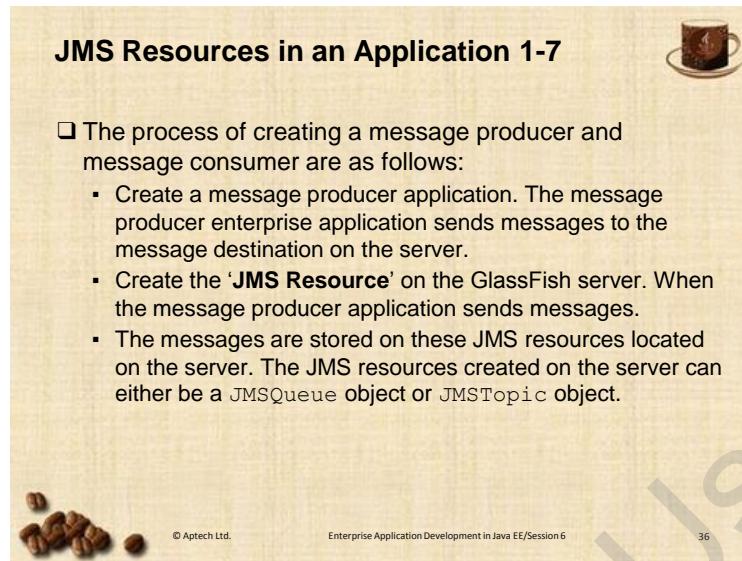
Use slide 34 to demonstrate creating a JMS resource in the application. The wizard as shown on slide 34 will appear when the developer adds a new file to an existing enterprise application. The enterprise application is **CreditCardValidation** to which a new file is being added. Choose JMS resource in the wizard.

Use slide 35 to demonstrate how the developer can choose the JMS resource to be created on the JMS application server.

In slide 35, a Queue type of JMS resource is being created. Highlight and discuss other resources that can be added to the application.

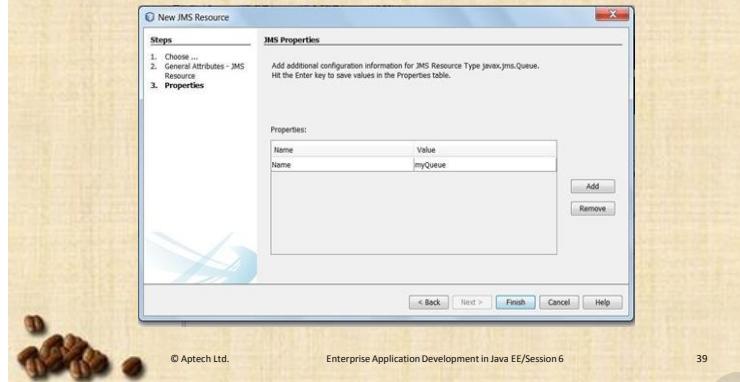
Slides 36 to 42

Let us understand the steps required to send a message.



JMS Resources in an Application 4-7

- Following figure shows how to provide a name to the JMS resource on the server:



JMS Resources in an Application 5-7

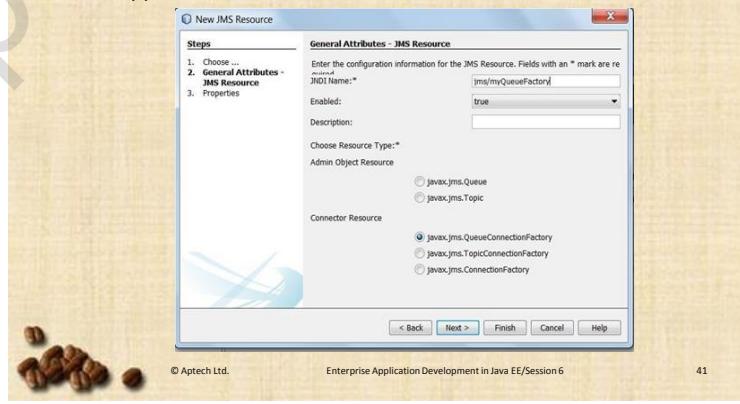
- The information about **myQueue** is added to the `glassfish-resources.xml` file in the Server Resources folder.

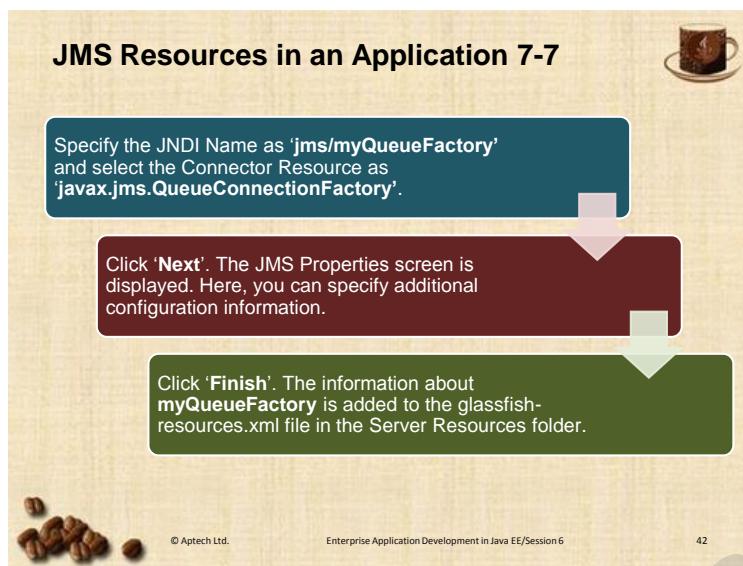
Creating ConnectionFactory

- The `ConnectionFactory` resource is required to make connections between the message producer module and the `Queue` object on the server.
- Right-click the project and select **New** → **Other** → **GlassFish JMS Resource** from the New File dialog box.
- Click **Next**.

JMS Resources in an Application 6-7

- Following figure shows how to create `QueueConnectionFactory` and `Connector` resources for the application:





Use slides 36 to 42 to explain the basic components required for sending a message in an application.

In order to send messages there should be a message producer application and a JMS resource which can act as a destination on the application server. The JMS resource can be a Queue object or a Topic based on the message communication model.

Use slide 37 to demonstrate how to create a JMS resource in an application. The application in this case is **MessageProducer-ejb**. Adding a new file to this application opens the wizard as shown in the slide.

Use slide 38 to demonstrate how to choose the type of JMS resource to be added to the application. In this slide, tell them that the developer has to specify a JNDI name for the resource and choose the type of the resource from the given set of options.

Use slide 39 to define the properties of the JMS resource. Every JMS resource has a name attribute. The JNDI name of the resource is assigned to this attribute.

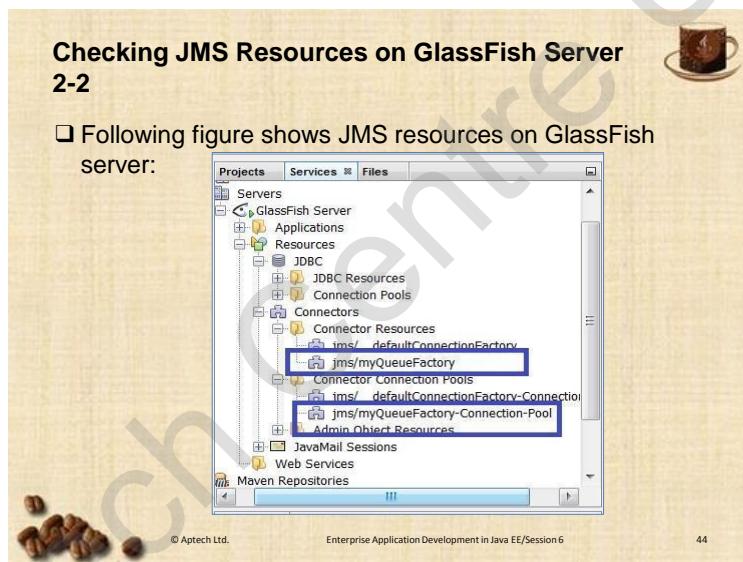
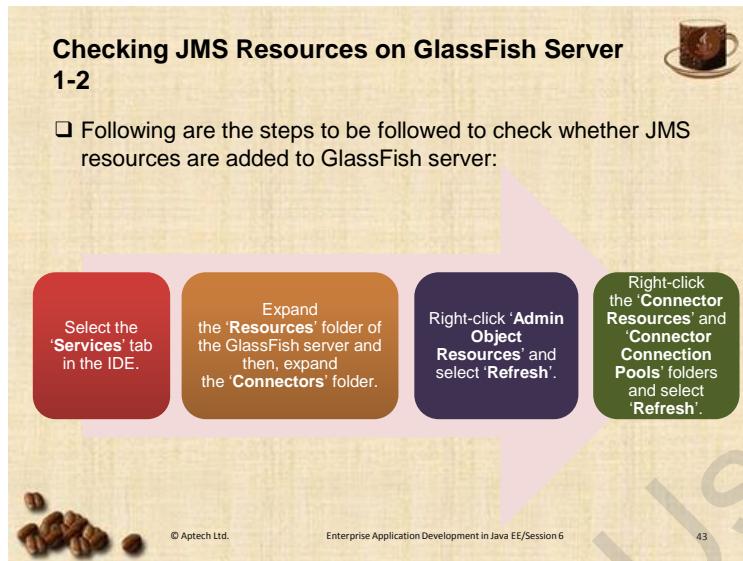
Use slide 40 to explain the process of establishing a Connection with the Destination object. The application client has to acquire a Connection object. The Connection object is created from a ConnectionFactory which is also a JMS resource. Since the process of creating a ConnectionFactory is similar to that of adding a Destination object, therefore the steps are also similar.

Use slide 41 to demonstrate how a developer can choose the correct options in the wizard to create a ConnectionFactory. In this case, a QueueConnectionFactory is being created.

Use slide 42 to explain the steps followed in configuring the QueueConnectionFactory. The steps are similar to those followed for creating a Destination on the application.

Slides 43 and 44

Let us check the resources created on the server.



Use slides 43 and 44 to explain the steps to be followed to check the resources created in the application.

To check whether the JMS resources are added to the server:

1. Select the 'Services' tab in the IDE.
2. Expand the 'Resources' folder of the GlassFish server and then, expand the 'Connectors' folder.
3. Right-click 'Admin Object Resources' and select 'Refresh'.
4. Right-click the 'Connector Resources' and 'Connector Connection Pools' folders and select 'Refresh'.

Use slide 44 to demonstrate the method of locating the resources created in the NetBeans IDE.

Slides 45 to 47

Let us understand how to define a MessageProducer class.

Creating Bean Objects in MessageProducer Application 1-3



Following are the steps to create a JSF managed bean which will act as message producer:

Right-click the project name and select **New** → **Other** → **JavaServer Faces** → **JSF ManagedBean** from the New File dialog box.

Click 'Next'. The New JSF ManagedBean dialog box displays on screen.

Specify the class name of the bean as '**MessageProducerBean**' and click 'Finish'. The ManagedBean is created.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 45

Creating Bean Objects in MessageProducer Application 2-3



Following code snippet shows the code for the managed bean created in **MessageProducer** application:

```
package mes;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class MessageProducerBean {

    /**
     * Creates a new instance of MessageProducerBean
     */
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 46

Creating Bean Objects in MessageProducer Application 3-3



```
private String message;

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}
public MessageProducerBean() {
}

}
```

A JSF ManagedBean is created to which the attribute message has been added. The getter and setter methods for this property have also been generated.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 47

Use slides 45 to 47 to explain how a MessageProducer class can be defined in the application. A JSF Managed bean is being created in this case. JSF stands for Java Server Faces. Message to be sent is generated in the bean class.

JSF is a specification used to develop user interfaces for component-based Web applications. The JSF Managed bean processes the data. The sample application is being defined such that certain data is received through the user interface and posted as message to the JMS destination.

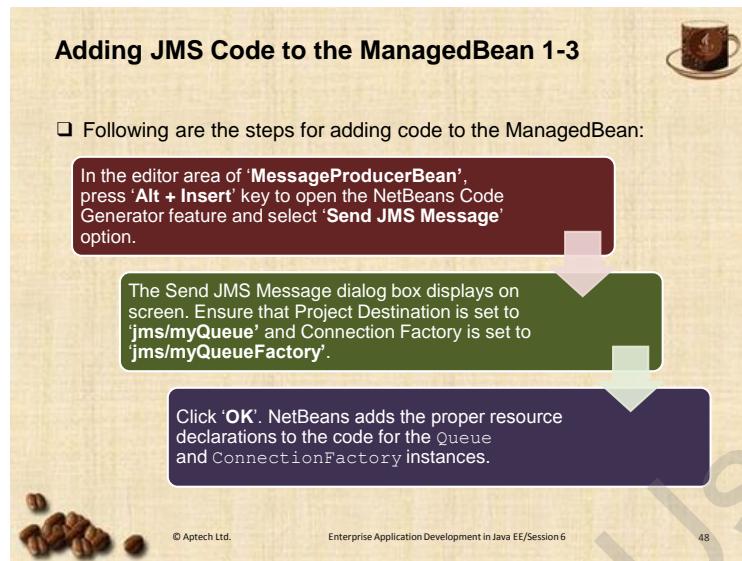
Use slides 46 and 47 to demonstrate the code created by the IDE when a JSF Managed Bean is created. The code shows an annotation `RequestScoped` which implies that the scope of the bean is only till the request lasts unlike the scope of the Singleton Session bean which lasts till the lifecycle of the application. The bean defined here is a Stateless Session bean, which is picked from the pool of beans when there is a request from the user interface and the bean is returned to the pool after the job is done.

Remember, here the bean is a client side bean, that is, a `MessageProducer` bean. This bean is used only when the client wants to send a message.

The `MessageProducer` bean is sending a String message in this bean. It is added as an instance variable to the JSF Managed Bean. Add getter and setter methods for this variable in the bean.

Slides 48 to 50

Let us understand how to add code to the MessageProducer bean.



Adding JMS Code to the ManagedBean 1-3

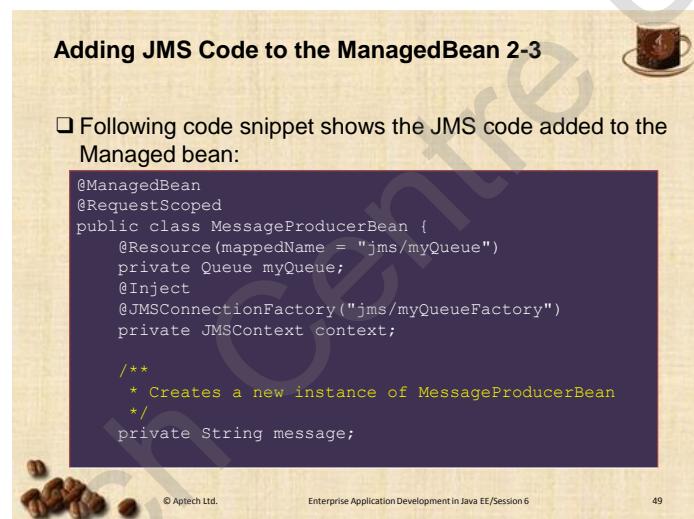
□ Following are the steps for adding code to the ManagedBean:

In the editor area of 'MessageProducerBean', press 'Alt + Insert' key to open the NetBeans Code Generator feature and select 'Send JMS Message' option.

The Send JMS Message dialog box displays on screen. Ensure that Project Destination is set to 'jms/myQueue' and Connection Factory is set to 'jms/myQueueFactory'.

Click 'OK'. NetBeans adds the proper resource declarations to the code for the Queue and ConnectionFactory instances.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 48



Adding JMS Code to the ManagedBean 2-3

□ Following code snippet shows the JMS code added to the Managed bean:

```
@ManagedBean
@RequestScoped
public class MessageProducerBean {
    @Resource(mappedName = "jms/myQueue")
    private Queue myQueue;
    @Inject
    @JMSConnectionFactory("jms/myQueueFactory")
    private JMSContext context;

    /**
     * Creates a new instance of MessageProducerBean
     */
    private String message;
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 49



Adding JMS Code to the ManagedBean 3-3

```
public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}
public MessageProducerBean() {
}

}
```

□ A `Queue` object has been added which is meant to be the destination of the message sent from the session bean.

□ A `JMSConnectionFactory` object is also added as a resource which enables the message sender to connect to the destination on the application server.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 50

Use slides 48 to 50 to demonstrate the code to send message from a MessageProducer bean.

Following are the steps to do so:

1. In the editor area of '**MessageProducerBean**', press '**Alt+Insert**' key to open the NetBeans Code Generator feature and select '**Send JMS Message**' option.
2. The Send JMS Message dialog box displays on the screen. Ensure that Project Destination is set to '**jms/myQueue**' and Connection Factory is set to '**jms/myQueueFactory**'.
3. Click '**OK**'. NetBeans adds the proper resource declarations to the code for the Queue and ConnectionFactory instances.

These steps will add code to the code editor as explained.

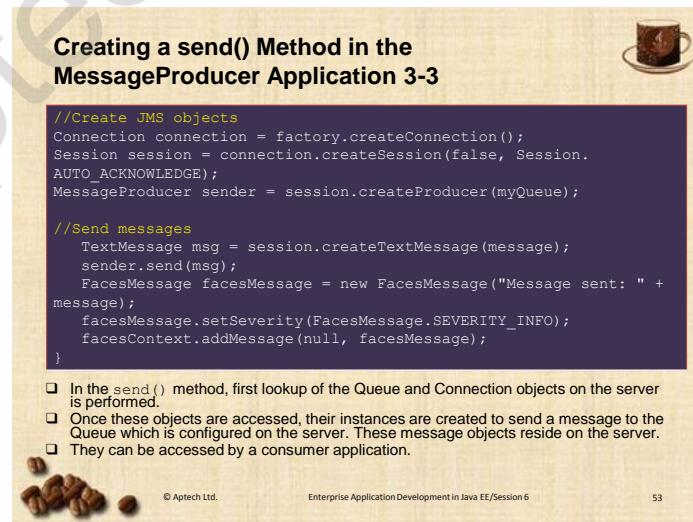
Use slides 49 and 50 to demonstrate the code added when developers try to send a message.

The code shows a resource added to the bean which is `myQueue`. Remember, we added this JMS resource to the enterprise application. We also added a `ConnectionFactory` resource through Admin Console earlier. These two components together provide the `JMSContext` for the message producer.

To send a JMS message, the message destination and connection to the message destination need to be configured. In the code snippet, a `Queue` object has been added which is meant to be the destination of the message sent from the session bean. A `JMSConnectionFactory` object is also added as a resource which enables the message sender to connect to the destination on the application server. When the session bean intends to send a message to the destination object, it establishes a connection with the server and sends the message onto the destination queue on the application server.

Slides 51 to 53

Let us understand the actual message sending process.



Use slides 51 to 53 to explain the message sending process.

Messages are sent through a `send()` method. The `send()` method requires a context to send the messages, for this, the `InitialContext` of the application is invoked which is set by the user interface of the application. The initial state of the user interface is retrieved.

The `Connection` object which is used for sending the message and the `Queue` object which will be the destination of the message are bound by their respective JNDI names. Instances of `Connection`, `Session`, and `MessageProducer` are created. Then, `send()` method of `MessageProducer` application is invoked to send the message. Since this message is picked from the user interface a Java Server Faces context is required.

The user interface will accept String from the application and send it as message to the destination through the bean.

Slides 54 and 55

Let us understand how to create a MessageConsumer application.

Creating a receive() Method in MessageConsumer Application 1-2



□ Create a **MessageConsumer** application. The managed bean within this application should have a **receive()** method to receive messages.

□ Following code snippet shows the code in the **receive()** method:

```
public class ReceiverBean implements javax.ejb.SessionBean
{
    InitialContext jndiContext;
    public String receiveMessage() {
        try{
            QueueConnectionFactory f = (QueueConnectionFactory)
                jndiContext.lookup("java:comp/env/jms/myQueueFactory");
            jndiContext.lookup("java:comp/env/jms/myQueue");
        }
    }
}
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 54

Creating a receive() Method in MessageConsumer Application 2-2



QueueConnection connect = factory.createQueueConneciton();
 QueueSession session = connect.createQueueSession(true, 0);

 QueueReceiver rec = session.createReceiver(myqueue);
 TextMessage textMsg = (TextMessage)rec.receive();
 connect.close();
 return textMsg.getText();
} catch(Exception e) {
 throws new EJBException(e);
}
}



© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 55

Use slides 54 and 55 to demonstrate the **receive ()** method of a **MessageConsumer**.

Developers have to create a **MessageConsumer** application similar to **MessageProducer**. The difference between a **MessageProducer** and **MessageConsumer** being, the message producer has a **send()** method and the message consumer has a **receive ()** method.

The **receive ()** method has to create the context for receiving the message by creating a **ConnectionFactory** and **Queue destination** instance. It has to then create a **Connection**, **Session**, and **MessageProducer** instances. The **receive ()** method is invoked with the **MessageProducer** instance. The **receive ()** method receives the message from the destination object defined.

Additional References:

To practice an example on JMS API, visit to the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/bnch.html>

Slide 56

Let us understand the process of message processing.



The slide has a light beige background with a faint watermark of coffee beans in the bottom left corner. At the top center, the title "Introduction to Message-Driven Bean 1-2" is displayed in a dark font. In the top right corner, there is a small icon of a coffee cup on a saucer. The main content area contains a section titled "Message-driven Bean" with a square bullet point. Below this, a bulleted list provides eight characteristics of message-driven beans:

- Is an enterprise bean used to process asynchronous messages received from application components.
- Is an EJB component that receives JMS message and other messages.
- Is a stateless, server-side component that is used for processing asynchronous messages delivered using JMS.
- Is invoked by the container and is responsible for processing messages.
- Does not have a remote or local business interface.
- Multiple instances of a message-driven bean can simultaneously process messages.
- Message-driven beans cannot be accessed through interfaces.
- Every message-driven bean has an `onMessage()` method to be executed.

At the bottom left of the slide, there is a small copyright notice: "© Aptech Ltd." and "Enterprise Application Development in Java EE/Session 6". At the bottom right, the page number "56" is visible.

Use slide 56 to explain the concept of Message-driven beans which are used on the server-side to process the messages received from message producers.

Message-driven beans are like Stateless Session beans, but they cannot be explicitly invoked through business interfaces. They can only be invoked through incoming messages.

Each message-driven bean has an `onMessage()` method. This method has all the actions to be carried out when a message is received.

Slide 57

Let us understand the characteristics of Message-driven beans.

Introduction to Message-Driven Bean 2-2

- ❑ Following are the characteristics of message-driven bean:
 - A message-driven bean is executed only upon receiving a message.
 - They are short lived and are invoked asynchronously.
 - Message-driven beans can access and update database, but do not represent the data in the database.
 - Message-driven beans are stateless and transaction aware.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 57

Use slide 57 to explain the characteristics of Message-driven beans.

- A message-driven bean is executed only upon receiving a message.
- They are short lived and are invoked asynchronously.
- Message-driven beans can access and update database, but do not represent the data in the database.
- Message-driven beans are stateless and transaction aware.

In-Class Question:

After you finish explaining the characteristics of Message-driven bean, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



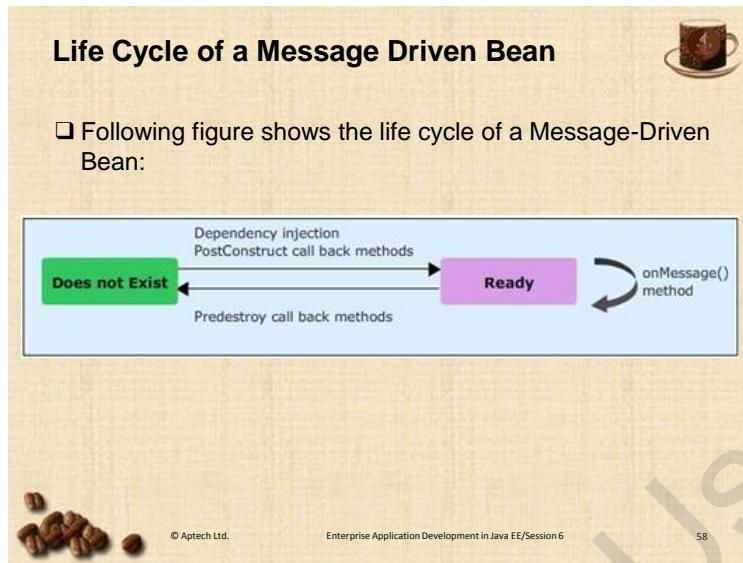
The Message-driven bean is similar to which Session bean?

Answer:

Stateless

Slide 58

Let us understand the life cycle of message-driven beans.



Use slide 58 to explain the life cycle of a message-driven bean. The life cycle of a message-driven bean is similar to that of Stateless Session bean. The bean initially exists in '**Does Not Exist**' state. It is instantiated into the container.

Once instantiated, required resources are injected into the bean and the life cycle callback methods are also invoked. The bean transforms from a Does not exist state to Ready state.

A bean in Ready state is invoked by the messages received from clients. On receiving a message the `onMessage()` method of the Message-driven bean is invoked to perform the required operation. After handling the received message, the Message-driven bean is in **Ready state** and can handle client messages.

Additional References:

To get more information on Message-driven bean, visit the following link:

<http://docs.oracle.com/cd/E19798-01/821-1841/gipkw/index.html>

Slide 59

Let us understand the callback methods in message-driven beans.

Lifecycle Event handlers

- ❑ Callback methods which are invoked when the message-driven bean transits from one state to another in the life cycle.
- ❑ Message-driven bean can have two types of callback methods:
 - `PostConstruct`
 - `PreDestroy`
- ❑ Callback methods should be appropriately annotated with `@PostConstruct` and `@PreDestroy`.

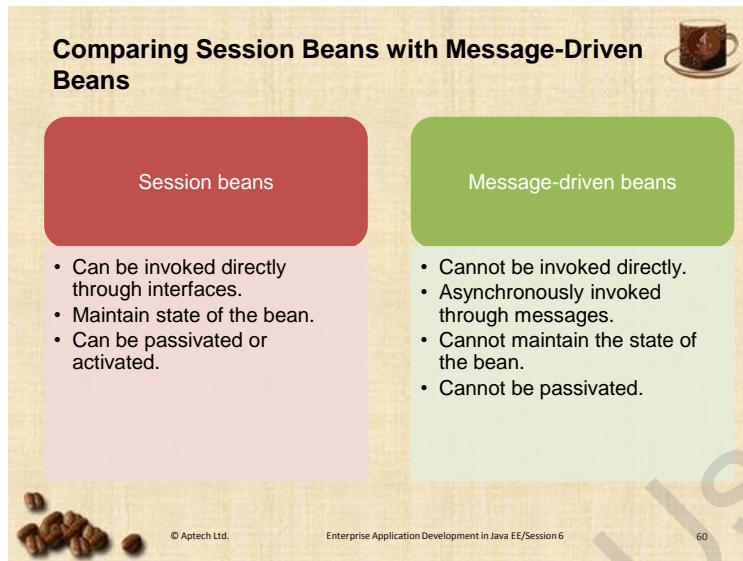
© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 59

Use slide 59 to explain the callback methods associated with the message-driven beans.

Like Stateless Session beans, message-driven beans cannot be passivated and activated. There are two types of callback methods associated with message-driven beans – PostConstruct and PreDestroy methods. PostConstruct methods are invoked when the message transits from **Does Not Exist** to **Method Ready** state. PreDestroy methods are invoked before the method transits from **Method Ready** to **Does Not Exist** state.

Slide 60

Let us compare Session beans with message-driven beans.



Use slide 60 to highlight the differences between Session beans and Message-driven beans.

- Session beans can be invoked directly through the interface provided by the developer, whereas Message-driven beans cannot be invoked.
- Message-driven beans are similar to stateless session beans as they do not maintain any conversational state of the application.
- Message-driven beans are asynchronously invoked by the messages.
- Message-driven beans like Stateless Session beans have only two states in their life cycle. They cannot be passivated like Stateful Session beans.
- Message-driven beans, like Stateless Session beans provide scalability in handling multiple clients.

Slide 61

Let us understand the process of creating and configuring Message-driven beans.

Creating and Configuring Message-Driven Beans 1-7

- ❑ Message-Driven Bean (MDB) is created like session beans in an enterprise application.
- ❑ MDB must implement `javax.jms.MessageListener` interface.
- ❑ They should have definition for `onMessage()` method.
- ❑ The implementation class of MDB should have a default constructor.

EJB 3.0, it is not compulsory to implement the `MessageDrivenBean` interface.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 61

Use slide 61 to explain the process of creating a Message-driven bean.

Message-driven beans can also be termed as message listeners because they act only when a message is received on the destination.

Message listeners asynchronously receive messages from the destinations. A message listener class must implement `javax.jms.MessageListener` interface. In earlier versions, a Message-driven bean implemented the `javax.ejb.MessageDrivenBean` interface. The `onMessage()` method accepts a `Message` object as a parameter and is invoked on receiving a message of parameter type.

However, in EJB 3.0, it is not compulsory to implement the `MessageDrivenBean` interface.

Slide 62

Let us understand the configuration code of a message listener.

Creating and Configuring Message-Driven Beans 2-7

Following code snippet shows the code used for creating a message listener for a consumer:

```
....  
JMSContext J = connectionFactory.createContext();  
JMSConsumer JC = J.createConsumer(destination);  
Listener L = new Listener();  
JC.setMessageListener(L);  
....
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 62

Use slide 62 to explain how a message listener can be defined for certain destination.

Explain the code snippet that sets a message listener object can be defined on a consumer using `setMessageListener()` method. Line 1 sets a context for message consumption. Line 2 defines message consumer on the message destination. Line 3 creates an instance of `MessageListener`. Line 4 sets the message listener on the `Destination` object.

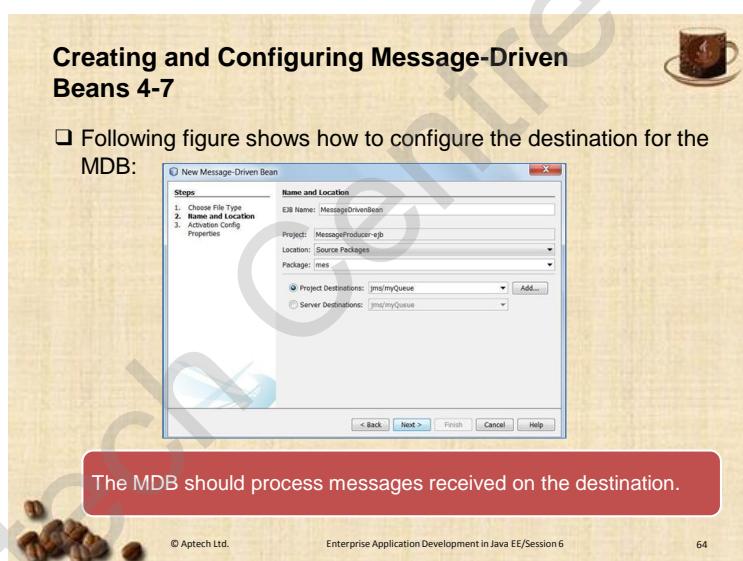
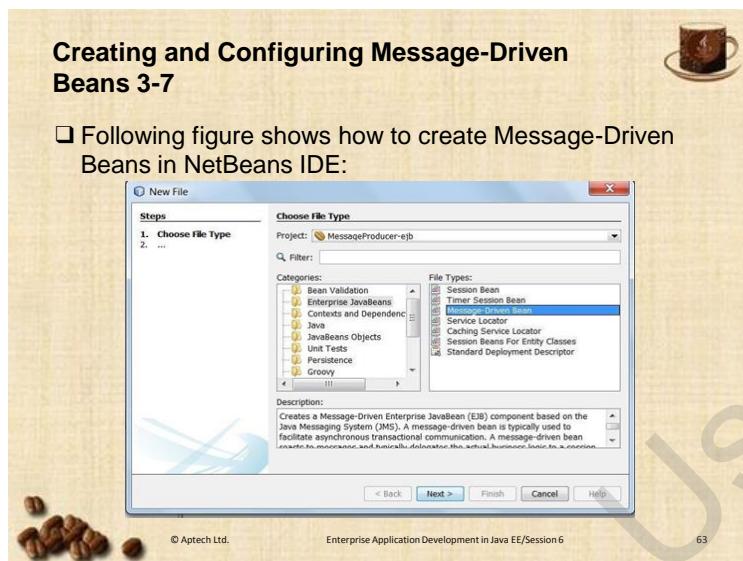
Additional References:

For more information on developing a message-driven bean using EJB 3.0, visit the following link:

<http://www.oracle.com/technetwork/middleware/ias/how-to-ejb30-mdb-093968.html>

Slides 63 and 64

Let us understand how to create a message-driven bean through NetBeans IDE.

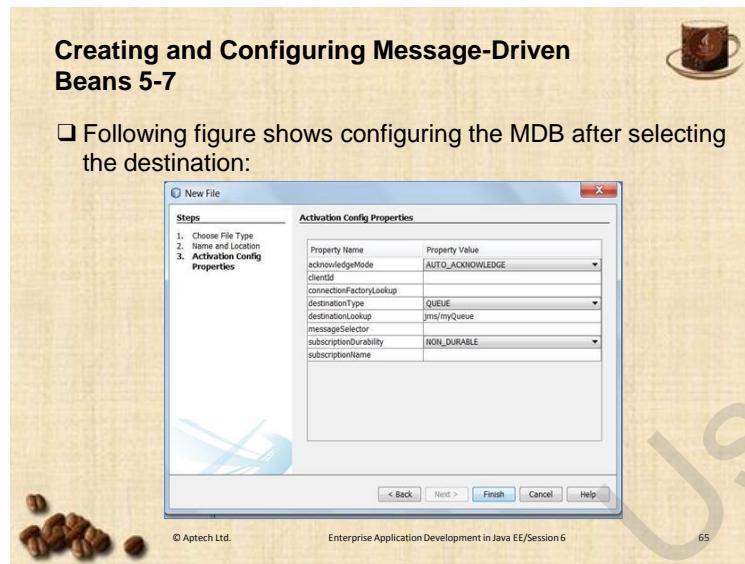


Use slides 63 and 64 to create a message-driven bean on the server. Make appropriate choices as demonstrated in the slide.

Use slide 64 to demonstrate how to choose the Destination object for which the current message-driven bean is defined. There is another option of Server Destination; this is used for a Destination deployed on the application server.

Slide 65

Let us understand the properties associated with the message-driven bean.



Use slide 65 to define the properties of handling a message by the message-driven bean. The strategy of delivering the acknowledgement is set to AUTO_ACKNOWLEDGE. The destination type, the destination's JNDI name and the subscription type of the messages is defined through these properties.

Additional References:

To prepare on message-driven bean, you can also implement the example as shown on the following link:

<https://docs.oracle.com/cd/E19575-01/819-3669/bnbpk/index.html>

Slides 66 and 67

Let us understand the code generated while creating and configuring a message-driven bean.

Creating and Configuring Message-Driven Beans 6-7



Following code snippet shows the code for `onMessage()` method of the MDB:

```
package mes;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName =
        "destinationType", propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName =
        "destinationLookup", propertyValue = "jms/myQueue")
})
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 66

Creating and Configuring Message-Driven Beans 7-7



```
public class MessageDrivenBean implements MessageListener
{
    public MessageDrivenBean() {
    }

    @Override
    public void onMessage(Message message) {
        ...
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 67

Use slides 66 and 67 to demonstrate the code created in the message-driven bean. Developers can add code in `onMessage()` method.

Additional References:

To get a complete example on creating an enterprise application using EJB 3.1, visit the following link:

<https://netbeans.org/kb/docs/javaee/javaee-entapp-ejb.html>

Slide 68

Let us summarize the session.

Summary

- ❑ JMS API can be used to create communication modules of enterprise applications. Apart from being part of an enterprise application, independent JMS applications can also be created.
- ❑ JMS primarily communicates through two messaging models, Publish-Subscribe model, and Point-to-point model.
- ❑ The prime focus of JMS API is to provide asynchronous communication.
- ❑ Communicating components can define message listeners on the message destinations.
- ❑ JMS API Programming model defines how different Java classes are used to implement the JMS application.
- ❑ Message-driven beans are defined in Java EE which are invoked on receiving a message on the message destination.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 6 68

Using slide 68, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

6.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concepts of Interceptors and Dependency injection.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 7 – Interceptors and Dependency Injection

7.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

7.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain aspect-oriented programming
- List the advantages of aspect-oriented programming
- Explain interceptors and their usage in enterprise applications
- Describe the different types of interceptor methods
- Explain how to define the interceptors and their levels of description
- Explain the creation of interceptor class
- Explain how to implement business method and lifecycle callback method interceptors
- Describe various aspects of context dependency and Injection
- Describe how to use CDI in enterprise applications

7.1.2 Teaching Skills

To teach this session successfully, you should be aware of what is aspect-oriented programming and how is it used in enterprise applications. You should be aware about the concept and working of interceptors and their use in the enterprise applications. Further, you should prepare yourself with the implementation of the interceptor classes, defining them on business methods, and lifecycle callback methods.

You should also be aware of Contexts and Dependency Injection (CDI). Also, prepare yourself on how Java EE uses CDI for performing resource injection in the enterprise applications.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- Explain aspect-oriented programming
- List the advantages of aspect-oriented programming
- Explain interceptors and their usage in enterprise applications
- Describe the different types of interceptor methods
- Explain how to define the interceptors and their levels of description
- Explain the creation of interceptor class
- Explain how to implement business method and lifecycle callback method interceptors
- Describe various aspects of context dependency and Injection
- Describe how to use CDI in enterprise applications

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 2

Tell them that in this session they will learn the concept of aspect-oriented programming used to separate concerns from the business code. Then, they will learn how to use interceptors in Java EE applications and their purpose in EJBs.

Further, they will learn the Context and Dependency Injection (CDI) with respect to the resources injected in the enterprise applications.

7.2 In-Class Explanations

Slides 3 to 6

Let us understand aspect-oriented programming.

Introduction to Aspect-Oriented Programming 1-4

Every business logic has some dependencies on its execution.
For example, the parameters passed to a business method must be validated, before they are processed further by the method.

In a software architecture design, the services provided to the business logics for their successful execution are termed as concerns.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 3

Introduction to Aspect-Oriented Programming 2-4

Following figure shows the concerns incorporated in the banking application:

```

void transfer(Account fromAcc, Account toAcc, int amount, User user,
Logger logger) throws Exception {
    logger.info("Transferring money.");
    transaction.begin();
    if (!isUserAuthorised(user, fromAcc)) {
        logger.info("User has no permission.");
        throw new UnauthorizedUserException();
    }
    if (fromAcc.getBalance() < amount) {
        logger.info("Insufficient funds.");
        throw new InsufficientFundsException();
    }
    fromAcc.withdraw(amount);
    toAcc.deposit(amount);
    transaction.commit();
    logger.info("Transaction successful.");
    transaction.close();
}

```

The concerns are interrelated to the business logics and are also referred to as business logic concerns.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 4

Introduction to Aspect-Oriented Programming 3-4



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 7

5

- ❑ To handle these types of concerns separately from the business logic, modern languages and frameworks provide support for Aspect-Oriented Programming (AOP).
- ❑ AOP allows the code developer to express cross-cutting concerns into stand-alone modules called aspects.
- ❑ The aspects intercept the request invocation executed for the object.
- ❑ Then, the aspects are injected into the functionality and finally, the request is delegated to the targeted object.
- ❑ Some of the common aspects include logging, transaction, security, connection, and so on.

Introduction to Aspect-Oriented Programming 4-4

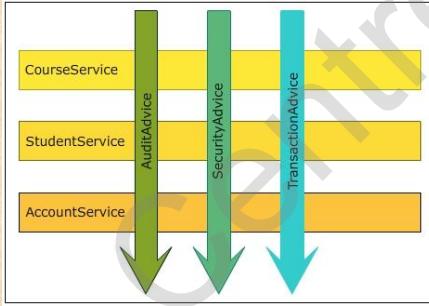


© Aptech Ltd.

Enterprise Application Development in Java EE/Session 7

6

- ❑ Following figure shows the aspects injecting into various business codes of the enterprise application:



Use slides 3 to 6 to explain aspect-oriented programming.

Applications require certain services for successful execution of the application. These services are also termed as concerns of the application. The services required for enterprise applications are namely, application logging, auditing, security, transactions, and so on.

These services are dependencies that every business logic has on its execution. For example, the parameters passed to a business method must be validated, before they are processed further by the method.

In a software architecture design, the services provided to the business logics for their successful execution are termed as concerns. The concerns are interrelated to the business logics and are also referred to as business logic concerns.

Aspect-Oriented Programming (AOP) defines how these services can be applied across different components of the application.

Use slide 4 to explain how concerns are implemented in the application code. The concerns shown in the code are related to application logging and transaction management. These concerns are applied across different application components.

Explain the figure as shown on slide 4 that represents the concerns incorporated in the banking application. Consider a banking application that has a business method, which is performing transferring of an amount from one account to another.

Now, consider if you have to incorporate a new concern such as security which would be span across multiple statements within the code, then, the business logic will be affected with the changes.

To handle these types of concerns separately from the business logic, modern languages and frameworks provide support for Aspect-Oriented Programming (AOP).

AOP allows the code developer to express cross-cutting concerns into stand-alone modules called aspects. The aspects intercept the request invocation executed for the object. Then, the aspects are injected into the functionality and finally, the request is delegated to the targeted object. Some of the common aspects include logging, transaction, security, connection, and so on.

Use slide 5 to explain the features of Aspect-Oriented Programming. The concerns of the application which have to be implemented across different components of the application are implemented as independent modules. Java EE also provides a framework to implement concerns across the application components.

Use slide 6 to explain how different aspects of the application are injected into the business code of the application. Explain the figure as shown on slide 6 that presents some aspects injecting into various business codes of the enterprise application.

The aspects shown in the application are **AuditAdvice**, **SecurityAdvice**, and **TransactionAdvice** for the application.

These aspects are implemented on the **AccountService**, **StudentService**, and **CourseService** components of the application.

Tips:

Earlier these aspects were implemented as part of the application code, but this implementation made the fine tuning of the aspects complicated hence current frameworks provide these concerns as independent modules.

In-Class Question:

After you finish explaining the aspect-oriented programming, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the use of AOP in designing enterprise applications?

Answer:

AOP allows the code developer to express cross-cutting concerns into stand-alone modules called aspects.

Slides 7 and 8

Let us understand the advantages of Aspect-Oriented Programming.



Advantages of Aspect-Oriented Programming 1-2



- Reusability**
 - By modularizing the code, it can be injected into multiple objects.
- Separation of concern**
 - Separation of concerns from the business logic makes the implementation independent of the concerns.
- Focus**
 - Developers can concentrate on aspects separately without merging them with the business logics.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7



Advantages of Aspect-Oriented Programming 2-2



On Java EE platform, EJB 3 supports AOP framework through Interceptors.

- Interceptors**
 - Interceptors are used to enable inclusion of aspects within the EJBs.
 - They provide the capability to intercept business methods and lifecycle callback methods.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7

Use slides 7 and 8 to explain the advantages of AOP. AOP allows reusability of application code and modularization of the concerns applying to all application components.

Separation of concerns from the application code enables the developers to focus on the business logic of the application.

Use slide 8 to explain how Java EE platform provides for implementing cross cutting aspects in the application. Java EE provides interceptors to implement the cross cutting aspects in the application.

Though interceptors and aspect-oriented programming are implemented differently, the purpose served by both the mechanisms is same. Java EE provides interceptors to implement functions such as application logging, transaction management, and security.

Slide 9

Let us understand interceptors in Java EE.

Interceptors 1-2

- ❑ Allow the developers to interpose EJB business methods.
- ❑ Add a wrapper code which is executed before or after the method is called.
- ❑ Are used to perform tasks such as logging information of bean method execution, auditing, and so on.
- ❑ Provide a fine grained control to the bean developer on the lifecycle methods of a bean.
- ❑ Used with Session beans and Message-driven beans.
- ❑ Defined through annotations or deployment descriptors.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 9

Use slide 9 to introduce the concept of interceptors.

Introduce the operations that can be performed by the interceptors. Interceptors allow the developers to interpose EJB business method. This means they allow the developer to add a wrapper code which is executed before or after the method is called. Interceptors are used to implement functions which are to be executed before the execution of business methods. They provide a wrapper code to the business methods in the application class. Interceptors can be used by developers to perform operations such as logging, implementing security mechanisms, implementing application auditing, and so on.

Interceptors can be invoked along with business methods and lifecycle callback methods. Interceptors can be used with Session beans and Message-driven beans as well. Developers can define an interceptor on the entire class or only a business method. This means the developers can apply interceptors on the class or methods. Developers can use annotations or deployment descriptors to map certain interceptors onto the business methods.

Interceptors are used in association with enterprise bean classes to allow the invocation of interceptor methods on the associated target class. On Java EE platform, Interceptors may be used with Session beans or Message-driven beans.

Tips:

Interceptors were first introduced as Interceptors 1.0 in EJB 3.0. Further, EJB 3.1 provides a generic interception framework in the form of the Interceptors 1.1 Specification.

Slide 10

Let us understand the scenarios in which interceptors can be used.

Interceptors 2-2

Following are some of the scenarios in which interceptors can be executed:

- Method parameters are to be validated, before they are passed to the bean method.
- Security checks are required to be performed, before executing the bean method.
- Logging or profiling operations are performed, during the bean method execution.
- Certain operations required to be performed, before or after class instantiation in the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 10

Use slide 10 to explain the scenarios in which an interceptor can be used.

These operations are supposed to be executed before the actual execution of business methods. Interceptors execute them and then, invoke the methods after successfully completing these operations.

Then, discuss scenario's where the interceptors can be executed as explained on slide 10.

Tips:

Interceptors may throw exceptions on which they are interposing. These exceptions must be caught by the 'catch' block and suppressed. Interceptors recover from exceptions by calling `InvocationContext.proceed()`.

Slide 11

Let us understand different types of interceptors that are used in enterprise applications.

Types of Interceptors

- ❑ Business method interceptors**
 - Also referred as 'AroundInvoke' interceptors.
 - Intercepts invocation of business methods in Session/Message-driven beans.
 - Annotated with `javax.interceptor.AroundInvoke` annotation which designate the business method as an interceptor method.
- ❑ Lifecycle callback interceptors**
 - Intercepts the invocation of lifecycle methods of the session beans or message-driven bean instances.
- ❑ Timeout callback interceptors**
 - Also referred as 'AroundTimeOut' methods.
 - Defined for EJB timer services.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 11

Use slide 11 to explain different types of interceptors which can be used in enterprise applications.

There are three types of interceptors – business method interceptors, Lifecycle callback interceptors, and Timeout callback interceptors. Business methods interceptors are invoked before invoking the business method of the application. These interceptors are annotated with `@AroundInvoke`. Business methods can be methods in a Session bean class or Message-driven bean class.

Lifecycle callback interceptors are invoked before invoking the lifecycle callback methods in the application. The lifecycle callback methods are associated with Session beans and Message-driven beans. Lifecycle callback interceptors are annotated with the callback instances when the method has to be invoked such as `PreDestroy`, `PostConstruct`, and so on.

Timeout event interceptors can be used in situations such as when the method is waiting for acquiring a lock and the event times out. In such situations timeout callback events are invoked and the timeout callback interceptors are invoked before the timeout callback event occurs.

Additional References:

To get information on interceptors, visit the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/gkeci.html>

Slide 12

Let us understand how developers can define interceptors.

Defining Interceptors

❑ Developers can define interceptors in one of the following two ways:

- As an interceptor method within a target class
 - Target class is the bean class.
 - Methods are annotated with `@javax.Interceptor.Interceptors` annotation.
 - Bean can either impose interceptor class on all methods or impose interceptor class on explicit methods.
- As a separate interceptor class
 - Interceptor class contain methods which can be invoked along with the lifecycle callback methods of the target class or business methods of the target class.
 - Only one interceptor class can be defined for a target class.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 12

Use slide 12 to explain how developers can define interceptors in applications. There are two ways in which an interceptor can be defined in an application – as a method in the target class or as an interceptor class defined on a target class.

When defined as a method, the interceptor method is placed within the target class for which it is defined. An interceptor method is annotated with `@javax.Interceptor.Interceptors`. If there are more than one interceptors intercepting a method then all the interceptors can be mentioned as a comma separated list in the annotation.

The interceptor can also be defined in a separate interceptor class. The interceptor class contain methods which can be invoked along with the lifecycle callback methods of the target class or business methods of the target class. There can be only one Interceptor class defined for a particular target class.

When an interceptor is defined on a class or a method, the developer has to define when the interceptor has to be invoked by the application. An interceptor can be invoked at various instances such as before the target method invocation, after completing the execution of the target method, before instantiating the target class, and so on.

Interceptor class has all the interceptor methods to be used in the application. There are four instances when an interceptor can be invoked – `AroundInvoke`, `AroundTimeout`, `PostConstruct`, and `PreDestroy`. All these methods in the target class are appropriately annotated.

Tips:

When an interceptor class is applied to a target class, the `AroundInvoke` interceptor is invoked before the invocation of every business method in the target class.

Slide 13

Let us introduce different types of interceptors invoked at different levels in the application.

Levels of Interceptors

☐ Interceptors are defined at different levels in the application:

- Default interceptors
- Class-level interceptors
- Method-level interceptors

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 13

Use slide 13 to explain interceptors invoked at different levels in the application.

A Default interceptor is an interceptor that is invoked whenever a business method is invoked on any bean within the application deployment. They can be bound only through a deployment descriptor and apply to all the bean classes in the application.

Class-level interceptors apply to all the methods defined within a bean class. They can be defined either through `@Interceptors` annotation or through deployment descriptors.

Method-level interceptors are defined to intercept only a single method in the application. This can also be defined either through annotation or through deployment descriptor.

Tips:

One or more interceptors can be applied to all EJBs within a deployment (default interceptors), to all methods of one EJB, or to a single method of an EJB. Interceptors can be applied through an annotation or by using an XML deployment descriptor.

Slide 14

Let us understand how default interceptors are defined.

Default Interceptors

- They are defined in the deployment descriptor.
- They are applicable across all the EJBs deployed in the application.
- Following code snippet shows ejb-jar.xml with default interceptors:

```
<ejb-jar
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd" version="3.1">
<assembly-descriptor>
  <interceptor-binding>
    <ejb-name> * </ejb-name>
    <interceptor-class>com.beans.MyInterceptor</interceptor-class>
  </interceptor-binding>
...
</assembly-descriptor>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 14

Use slide 14 to explain how default deployment descriptors are applied in the application.

They are either defined in `ejb-jar.xml` or other deployment descriptor which is defined by the developer. The deployment descriptor defined by the developer is also an `.xml` file. The interceptors are bound in the deployment descriptor through `<interceptor-binding>` element.

Then, explain the given code snippet on the slide that defines how to use a default descriptor in the application. Tell them that the code snippet shows how an interceptor is defined on all the bean classes of the application. The default interceptors are invoked, before every method invocation in the application. The wildcard character '*' in the `<ejb-name>` element applies the interceptor to every EJB deployed in the application.

Slide 15

Let us understand how to define class-level interceptors.

Class-Level Interceptors

- ❑ They are defined on all methods of bean class.
- ❑ They are used to define the interceptors for all methods in the class using `@Interceptors` annotation.
- ❑ Following code snippet shows the usage of class-level interceptors:

```
@Stateless
@Interceptors(value=com.beans.MyInterceptor.class)
public class ApplicationBean {
    ...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 15

Use slide 15 to explain class-level interceptors.

Class-level interceptors are defined on the entire class, that is, the interceptors are invoked when any method of the target class is invoked. Class-level interceptors are defined either through `@Interceptors` annotation or through deployment descriptor.

Then, explain the code snippet that shows an interceptor `@Interceptors` annotation declared at the class-level intercepts all the methods of an `ApplicationBean` class.

Tips:

By default, the `@Interceptors` annotation is applied to all the method of EJB class. However, if you want that the interceptor should not be applied to a particular method, then you can use the `@ExcludeClassInterceptors` annotation with the method. The `@ExcludeClassInterceptors` annotation has no arguments and hence, will not use any interceptors declared at the class-level.

The following code snippet shows the use of `@ExcludeClassInterceptors` annotation:

```
@ExcludeClassInterceptors
public void register(String name, String address) {
    ...
}
```

Slide 16

Let us understand method-level interceptors.

Method-Level Interceptors

- ❑ They are defined to intercept a method invocation in the bean class.
- ❑ `@Interceptors` annotation is used to define method-level interceptors.
- ❑ Following code snippet demonstrates the usage of method-level interceptors:

```
...  
@Interceptors({AccountsConfirmInterceptor.class})  
public void sendBookingConfirmationMessage(long  
orderId)  
{  
    ...  
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 16

Use slide 16 to explain how developers can define method-level interceptors.

Method-level interceptors are defined to intercept only a method call in the EJB.

Method-level interceptors are defined by annotating the method with `@Interceptors` annotation. They can also be defined through the deployment descriptors. In the given slide, the method `sendBookingConfirmationMessage ()` is intercepted by interceptors defined in `AccountsConfirmInterceptor` class.

In-Class Question:

After you finish explaining method-level annotation, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What are the steps to apply interceptors at the method-level?

Answer:

The steps are:

- Define an interceptor
- Annotate the method with the `@Interceptors` annotations

Slide 17

Let us understand how to configure an interceptor class.

Configuring an Interceptor Class

Every Interceptor class is associated with a Stateless or Stateful session bean.

Following are the steps in configuring an interceptor class:

- Create an interceptor class as a simple POJO class.
- Define an interceptor method within the class.
- Designate the method with `@AroundInvoke` annotation or lifecycle annotation.
- Associate the interceptor class with EJB Session bean.

Use slide 17 to explain how to configure an interceptor class.

An interceptor class is defined like any other Java class. It only has the interceptor methods such as `AroundInvoke`, `AroundConstruct`, and `PostConstruct` methods. Each of these methods are appropriately annotated.

An interceptor class can be applied on a target Stateless Session bean or Stateful Session bean class.

Following are the steps to configure an interceptor class:

1. Create an Interceptor class which is a simple POJO class in Java.
2. Define a method in the Interceptor class which can be business method or lifecycle method.
3. Designate the method as the interceptor method with `@AroundInvoke` annotation or lifecycle annotations.
4. Apply or associate the defined interceptor class to the EJB session bean.

Additional References:

You can create an interceptor class and apply it to an EJB class by visiting the following link:

<http://www.javacodegeeks.com/2013/07/java-ee-ejb-interceptors-tutorial-and-example.html>

Slides 18 to 23

Let us understand the process of creating a method-level interceptor.

Creating Business Method Interceptor 1-6



- ❑ javax.interceptor.AroundInvoke annotation is used to create business method interceptors.
- ❑ Following code snippet shows the signature of the business interceptor method annotated with @AroundInvoke annotation:

```
@AroundInvoke  
Object <METHOD_NAME>  
(javax.Interceptor.InvocationContext) throws Exception  
{  
    ...  
}
```

- ❑ The method designates as the interceptor method.
- ❑ The developer can use only one @AroundInvoke annotation per class.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 18

Creating Business Method Interceptor 2-6



- ❑ The advantage of interceptors is that they provide the developer a way to add functionalities on the business methods without modifying the method code.
- ❑ Interceptor methods can be declared public, private, protected, or package level access.
- ❑ They cannot be declared final and static.
- ❑ They may throw runtime exceptions.
- ❑ They are invoked with InvocationContext object as a parameter.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 19

Creating Business Method Interceptor 3-6



- ❑ **InvocationContext object**
 - Is the generic representation of the business method for which the interceptor is defined.
 - Includes information such as target bean instance on which the interceptor is invoked, parameters with which the target bean is invoked, and so on.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 20

Creating Business Method Interceptor 4-6



- ❑ Following are some of the methods of `InvocationContext`:

- `Object getBean()`: Returns a reference to the bean on which the method is invoked.
- `Method getMethod()`: Returns a reference to the invoked method.
- `Object[] getParameters()`: Returns the parameters passed to the method.
- `void setParameters(Object[] parameters)`: Sets the parameters of the object.
- `Map getContextData()`: Get contextual data that can be shared in a chain.
- `Object proceed()`: Proceeds to the next interceptor in the chain or the business method, if it is the last interceptor.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 7

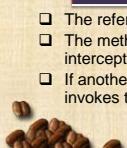
21

Creating Business Method Interceptor 5-6



- ❑ Following code snippet demonstrates the definition of an interceptor class:

```
...
public class MyInterceptor {
    @AroundInvoke
    public Object businessIntercept(InvocationContext ctx)
        throws Exception {
        Object result = null;
        System.out.println("perform intercepting operations");
        try {
            result = ctx.proceed();
        } finally {
        }
    }
}
```



© Aptech Ltd.

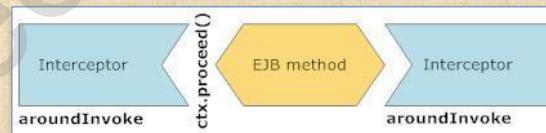
Enterprise Application Development in Java EE/Session 7

22

Creating Business Method Interceptor 6-6



- ❑ Following figure demonstrates wrapping business method with a chain of interceptors:



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 7

23

Use slides 18 to 23 to explain the process of creating a business method interceptor.

A business method is annotated with `@AroundInvoke`. This interceptor can be created within the target class or as a separate interceptor class.

The `@AroundInvoke` wraps around the call to your business method and is actually invoked in the same Java call stack and in the same transaction and security context as the bean method it is intercepting.

Then, explain the code snippet to explain the signature of the business interceptor method annotated with `@AroundInvoke` annotation. Tell them that the developer can use only one `@AroundInvoke` annotation per class.

Use slide 19 to explain the features of creating a business method interceptor.

The interceptor method can be declared with `public`, `private`, `protected`, or `package` level access. However, the method cannot be declared as `final` or `static`. The interceptors may throw runtime exceptions.

All interceptors have to be invoked in an `InvocationContext`. The `InvocationContext` of the interceptor should same as that of the business method on which it is being invoked.

Use slide 20 to explain why an interceptor requires an `InvocationContext`. An `InvocationContext` provides an execution environment for business method of the application.

The interceptor method is invoked with an `InvocationContext` object as a parameter. The `InvocationContext` object is the generic representation of the business method for which the interceptor is defined. The `InvocationContext` object includes information such as target bean instance on which the interceptor is invoked, parameters with which the target bean is invoked, and so on. The security and transaction context of the `@AroundInvoke` interceptor is same as that of the target method.

Use slide 21 to explain the methods defined in the `InvocationContext` class.

- `Object getBean()`: Returns a reference to the bean on which the method is invoked.
- `Method getMethod()`: Returns a reference to the invoked method.
- `Object[] getParameters()`: Returns the parameters passed to the method.
- `void set Parameters(Object[] parameters)`: Sets the parameters of the object.
- `Map getContextData()`: Gets contextual data that can be shared in a chain.
- `Object proceed()`: Proceeds to the next interceptor in the chain or the business method, if it is the last interceptor.

Use slide 22 to explain the process of defining an interceptor class in the application. In the slide, a `MyInterceptor` class has been created. This class has all the interceptors required.

The code shows `@AroundInvoke` interceptor being defined. When this class is applied as an interceptor class on the target method, the `AroundInvoke` interceptor is invoked on invocation of every business method in the target class.

If the application requires other interceptors also, it can be defined in the Interceptor class. A `proceed()` method is invoked by the `InvocationContext` after the interceptor completes execution.

Use slide 23 to graphically explain how an interceptor functions around a business method. The image shows an `AroundInvoke` interceptor wrapped around the EJB method.

The interceptor method executes first, after completing the execution of the interceptor, a `proceed()` method is invoked by the interceptor. This method signifies a clearance for the execution of the business method.

If there is an exception occurring in the interceptor then, the `proceed()` method is not executed.

In-Class Question:

After you finish explaining the process of creating a business method interceptor, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which object is passed as a parameter to the interceptor method?

Answer:

The interceptor method is invoked with an `InvocationContext` object as a parameter.

Slides 24 to 28

Let us understand the methods of applying interceptors.

Applying Interceptor 1-5



Developer can apply interceptors to the bean class by configuring the deployment descriptor.

Developers can also use `@Interceptors` annotation to apply the interceptor.

Following is the syntax for applying the interceptors:

```
@Interceptors({Interceptor1.class,Interceptor2.class})
targetMethod()
.
.
}
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 24

Applying Interceptor 2-5



Following code snippet shows the code of a target class which uses interceptor:

```
.
.
import javax.interceptor.AroundInvoke;
import javax.interceptor.Interceptors;
import javax.interceptor.InvocationContext;

@Stateless
@LocalBean
public class ExampleBean {
    @Interceptors(value = interceptors.MyInterceptor.class)
    public void sayHello(){
        System.out.println("SayHello");
    }
}
```

The code shows the target class on which the interceptor class is defined.

The `BusinessIntercept()` method is invoked before any method of the `ExampleBean` class is invoked.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 25

Applying Interceptor 3-5



Following code snippet shows the invocation of a bean method on which interceptor is defined:

```
package Beans;
public class InterceptorDemo {
    public static void main(String[] args) {
        ExampleBean E = new ExampleBean();
        E.sayHello();
    }
}
```

The code instantiates the `EmployeeBean` class and invokes the method `sayHello()` on it.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 26

Applying Interceptor 4-5



While defining the interceptor as an interceptor class, it must meet the following requirements:

- The interceptor class must have a default public constructor.
- The interceptor classes should be defined with `@Interceptors` annotation.
- When a target class has multiple interceptor classes defined for it, the order of Interceptor invocation should be defined through annotations.
- The order of invoking the interceptors in case of multiple interceptors can be defined in the deployment descriptor of the application also.
- The order defined in the deployment descriptor overrides the order defined through annotations.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 27

Applying Interceptor 5-5



Following code snippet shows the code to specify the interceptors in the deployment descriptor for a particular method:

```
...
<interceptor-binding>
<target-name> Target bean class </target-name>
<interceptor-class>First interceptor class </interceptor-class>
<interceptor-class>Second interceptor class</interceptor-class>
<interceptor-class>Third interceptor class</interceptor-class>
<method-name>Method to be intercepted</method-name>
</interceptor-binding>
...

```

The order in which the interceptors are invoked is same as the order in which these interceptors are specified in the deployment descriptor.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 28

Use slides 24 to 28 to explain different methods of applying interceptors on bean methods and bean classes.

Interceptors can be applied through deployment descriptor or through `@Interceptors` annotation.

In deployment descriptors, interceptors are mapped onto target classes through XML tags. Through annotations interceptors are defined through `@Interceptors` annotation.

Then, explain the syntax of applying the interceptor to the target method.

Use slide 25 to explain how the interceptor is applied on the target class with the help of code snippet.

In slide 25, the `@Interceptors` annotation has the value of the Interceptor class defined earlier. Whenever the business method `sayHello()` is executed, the corresponding `AroundInvoke` method in the interceptor class is invoked.

Use slide 26 to explain how the bean method is invoked. Before invoking the business method, the container invokes the interceptor. The slide shows the code of the target class on which the interceptor class is defined. The `businessIntercept()` method is invoked before any method of the `ExampleBean` class is invoked.

Use slide 27 to explain the programming rules of interceptors.

- The interceptor class must have a default public constructor.
- The interceptor classes should be defined with `@Interceptors` annotation.
- When a target class has multiple interceptor classes defined for it, the order of interceptor invocation should be defined through annotations.
- The order of invoking the interceptors in case of multiple interceptors can be defined in the deployment descriptor of the application also. The order defined in the deployment descriptor overrides the order defined through annotations.

Use slide 28 to explain the process of binding multiple interceptors with a target class through the deployment descriptor.

The code shows the Target bean class is bound with three interceptor classes. If these classes have to intercept a specific method, then the tag `<method-name>` can be used. The order in which the interceptors are invoked is same as the order in which these interceptors are specified in the deployment descriptor. The `<method-name>` element includes the method for which the interceptors will be executed in the specified order.

Slide 29

Let us understand the life cycle of an interceptor.

Life Cycle of an Interceptor

- ❑ Life cycle of an interceptor is the same as the target class or target method.
- ❑ When an instance of target class is created, then the interceptor class is also instantiated for the target class.
- ❑ In case of multiple interceptor classes, then an instance of each interceptor class is created.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 29

Use slide 29 to explain the lifecycle of an interceptor.

The life cycle of an interceptor is tied up with that of the target method or target class. When the interceptor is implemented as a class, it is instantiated when the target class is instantiated and when methods of the target class are invoked corresponding interceptors are invoked.

If the target class uses multiple interceptors, then in that case, the object of each applied interceptor class is instantiated, before the business methods are invoked on the target bean class instance.

All the interceptor classes must be instantiated, before the invocation of the `PostConstruct` callback method. Similarly, the `PreDestroy` callback method is invoked before the instances of target class and interceptor class are destroyed.

Slide 30

Let us understand lifecycle callback interceptors.

Lifecycle Callback Interceptors 1-3

- ❑ Lifecycle callback events occur whenever the bean object changes from Does Not Exist state to Ready state or when the bean object transits from activated state to passivated state, and so on.
- ❑ The lifecycle callback events are handled through the lifecycle callback methods.
- ❑ Following are the annotations related to the lifecycle methods of the bean class:
 - javax.interceptor.AroundConstruct
 - javax.annotation.PostConstruct
 - javax.annotation.PreDestroy

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 30

Use slide 30 to explain lifecycle callback interceptors. These interceptors are invoked when an object or a bean in the application transits from one state of lifecycle to another.

For instance, a Stateless Session bean may toggle between '**'Does Not Exist'**' and '**'Method Ready'**' states.

A Stateful Session bean may transit from '**'Does Not Exist'**' and '**'Method Ready'**' State and vice versa. Similarly it may also transit from '**'Method Ready'**' to '**'Passivated'**' state and vice versa.

Interceptors for these transitions can be defined through annotations.

- `javax.interceptor.AroundConstruct` – This annotation is used when the developer intends to invoke the interceptor method during a lifecycle event. When an enterprise bean is instantiated, if the developer intends to invoke an interceptor on it, then the `@AroundConstruct` annotation can be used.
- `javax.annotation.PostConstruct` – Interceptor methods prefixed with the annotation `@PostConstruct` are invoked after the bean object is instantiated into the container. These methods are used to perform the tasks required after the bean object is instantiated by the container. They are invoked after the lifecycle event of instantiation.
- Interceptor methods annotated with `@ PreDestroy` annotation are invoked when the bean object has to be removed from the container. These methods perform tasks such as deallocating the resources allocated for the bean and so on.

Slide 31

Let us understand the lifecycle callback interceptors for Stateful Session beans.

Lifecycle Callback Interceptors 2-3

- ❑ Apart from this, Stateful Session bean also have methods annotated with `@PrePassivate` and `@PostActivate`.
- ❑ The syntax for declaring the lifecycle interceptor methods is as follows:
 - `@callback-annotation`
 - `void method-name(InvocationContext ctx);`
- ❑ The method defined in the interceptor class will be annotated with the callback annotation.
- ❑ The return value of these method is void, as EJB callback methods do not return any value.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 31

Use slide 31 to explain the interceptor annotations used with Stateful Session beans in addition to those used in case of Stateless Session beans.

Interceptor methods do not return anything, their return type is always void and they accept the `InvocationContext` of the target method as a parameter.

Apart from this, Stateful Session bean also have methods annotated with `@PrePassivate` and `@PostActivate`.

- `@PrePassivate` annotation is used to annotate interceptors defined on Stateful session beans. It is invoked before the bean is passivated.
- `@PostActivate` annotation is also used to annotate interceptors defined on the Stateful session beans. It is invoked when the bean is removed from a passivated state.

If the lifecycle callback interceptors are defined within the target class they are annotated with appropriate annotations. If the interceptors are defined in an interceptor class then the interceptors have an `InvocationContext` parameter.

The lifecycle callback interceptors cannot be static or final. These interceptors may throw runtime exceptions.

Slide 32

Let us understand how lifecycle callback methods can be intercepted for Stateful Session bean.

Following code snippet shows an interceptor class for a Stateful Session bean with lifecycle callback interceptor methods:

```

public class MyStatefulSessionBeanInterceptor {
    protected void myInterceptorMethod (InvocationContext ctx) {
        ...
        ctx.proceed();
        ...
    }
    @PostConstruct
    @PostActivate
    protected void myPostConstructInterceptorMethod(InvocationContext ctx) {
        ...   ctx.proceed();   ...
    }
    @PrePassivate
    protected void myPrePassivateInterceptorMethod (InvocationContext ctx)
    {
        ...
        ctx.proceed();
        ...
    }
}

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 32

Use slide 32 to explain how lifecycle callback interceptor method is defined for Stateful Session bean.

In the given code snippet, the `myPrePassivateInterceptorMethod()` is defined as the lifecycle callback interceptor method for the pre-passive lifecycle event. Then, the method `myPostConstructInterceptorMethod()` is defined to handle the `PostConstruct` and `PostActivate` lifecycle events.

Slide 33

Let us understand Timeout interceptors.

Timeout Interceptors

- ❑ Timeout interceptors are invoked when a timeout event occurs on an object.
- ❑ `@AroundTimeOut` annotation is prefixed for such interceptor methods.
- ❑ Following is the syntax for using `@AroundTimeOut` annotation:

```
@AroundTimeOut
public Object InterceptorMethod() {
    . . .
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 33

Use slide 33 to explain Timeout interceptors.

Timeout interceptors are invoked in the event of a timer expiry. These interceptors are annotated with `@AroundTimeOut` event.

These interceptors can also be defined either in the target class or in an interceptor class.

This interceptor can be used to respond to timeout events, while the component is trying to establish connection with a remote application component. It can also be used when component is trying to acquire a lock or when the component is trying to access the database.

Then, explain the syntax to define the `@AroundTimeOut`.

Like other interceptors, the timeout event interceptors cannot be `static` or `final`. A timeout event interceptor can access all the resources accessed by target timeout method and has the same security constraints and transaction context as the target method.

Multiple timeout event interceptors can be used on the target methods, where the order of interceptors is specified through `@Interceptors` annotation or through the deployment descriptor.

Additional References:

To get more information on interceptors, visit the following link:
<http://docs.oracle.com/javaee/6/tutorial/doc/gkedm.html>

Slide 34

Let us understand interceptor chaining.

Interceptor Chaining 1-2

- ❑ A bean class can have multiple interceptors defined on it.
- ❑ The order of executing the interceptors has to be defined in the deployment descriptor.
- ❑ There are certain rules to determine the order for executing multiple interceptors.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 34

Using slide 34, explain interceptor chaining. A target class may have multiple interceptors defined on it. There may be two situations which lead to interceptor chaining. The execution of one interceptor might trigger the invocation of another one, and further the execution of the second interceptor may trigger the execution of another interceptor.

Sometimes, when multiple interceptors are defined on a target method or target class and it is essential that all the interceptors are executed. For this, the developer has to define the order in which these interceptors have to be invoked.

Slide 35

Let us understand the rules for defining the order of multiple interceptors.

Interceptor Chaining 2-2

Following are the rules which apply to interceptor chaining:

- The target class has a set of default interceptors which are defined in the deployment descriptor.
- The order of execution is defined through `@Interceptors` annotations.
- When interceptors are defined as a hierarchy of classes, the interceptor invocation starts from the super class.
- Interceptors can be associated with a priority through `javax.annotation.priority` annotation.
- Interceptors annotated with `@AroundConstruct` are executed before the target method.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 35

Use slide 35 to explain the rules which apply to interceptor chaining in the application.

- There are a set of default interceptors which are determined on a target class or bean, these are executed first. Default interceptors are specified in the deployment descriptor. These default interceptors are executed in the order in which they are specified in the deployment descriptor.
- The order of the interceptors can also be defined with the `@Interceptors` annotation. The order of execution is the order in which they are specified with the annotations. Priorities assigned to the interceptors are not applicable to the `@Interceptors` annotation.
- When interceptors are defined as classes, hierarchy is applicable among these classes. In such a scenario, the super class interceptors are executed first and then, the subclass interceptors.
- `javax.annotation.priority` annotation can be used to prioritize the interceptors. This annotation can be used to set priority to the interceptor methods.
- For interceptors annotated with `@AroundConstruct` annotation, the interceptors are invoked first and then, the constructor of the target bean class.

Additional References:

To prepare on the interceptors example, visit the following links:

<http://docs.oracle.com/javaee/6/tutorial/doc/gkeci.html>

<https://weblogs.java.net/blog/2006/01/25/interceptors-ejb-3>

Slide 36

Let us understand the concept of Context and Dependency Injection (CDI).

Context and Dependency Injection (CDI) 1-3

- ❑ CDI is a standard used to define the structure of the application.
- ❑ CDI enables integration of various components of the application in a loosely coupled manner.
- ❑ CDI allows EJBs to be used as managed beans for JSF applications.
 - This helps the Web tier to interact directly with the business and persistence tiers.
- ❑ CDI is also used to resolve dependencies among managed beans.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 36

Use slide 36 to explain the concept of contexts and dependency injection.

The Java EE 6 platform introduced Context and Dependency Injection (CDI) that helps to tie the Web tier and the business tier of the Java EE platform.

CDI is a set of services that make it easy for developers to use enterprise beans along with JSF technology in Web applications. CDI allows the EJBs to be used as managed beans for JSF applications. This helps the Web tier to interact directly with the business and persistence tiers.

CDI provides for a loosely coupled architecture of application. It allows beans and resources to be injected into the application as per requirement. It provides @Inject annotation through which developers can inject other components or beans into current application component.

CDI enables usage of managed beans for JSF applications. JSF applications are Web-based applications in which EJB components can be used to implement business logic of the application. CDI is responsible for resolving the dependencies on the managed beans.

Tips:

CDI enables your objects to have dependencies provided to them automatically, instead of providing them as parameters.

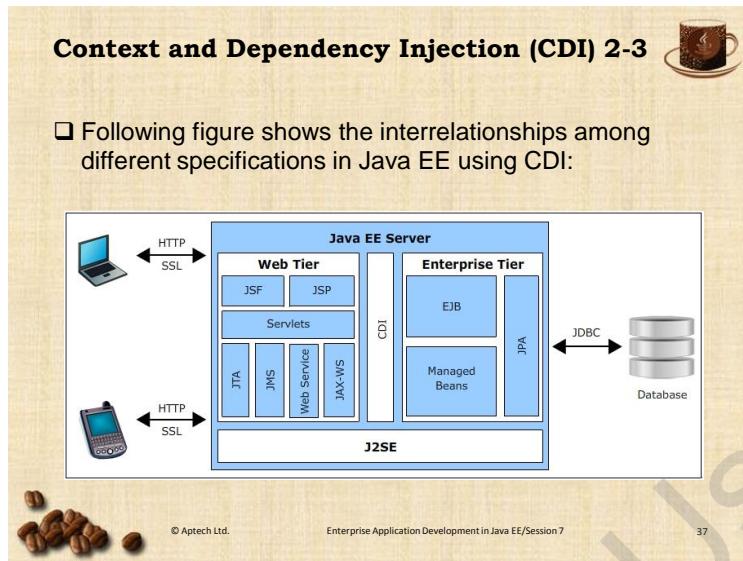
For example, consider a following example of Servlet class, MyServlet which need a class named Greeting to print the response on the page:

```
@WebServlet("/Myservlet")
public class MyServlet extends HttpServlet {
    // Dependent class
    private Greeting greet;
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        response.getWriter().write(greet.display());
    }
}
```

With CDI, the Servlet class can declare its dependency on a Greeting instance and have it injected automatically by the CDI runtime.

Slide 37

Let us understand the architecture supported by CDI in Java EE.



Use slide 37 to explain inter relationships among different components of the application using CDI.

On slide 37, figure shows the Java EE server comprising a Web tier and an Enterprise tier. CDI lies between the Web tier and Enterprise tier, which implies that enterprise components can be injected into Web applications through CDI. The figure demonstrates Java EE platform which depend on CDI for injecting the required dependencies in the application. The components such as Servlets, EJBs, JSP, and JSF depend on CDI for injecting the dependencies on managed beans in the application. The service of CDI is implemented through the container. Apart from the managed beans, other Java EE resources are also injected into the application through CDI.

Tips:

The major loose coupling achieved from CDI is as follows:

- Decouples the server and the client by means of well-defined types and qualifiers. This helps the server components to make changes.
- Decouples the lifecycles of tied components by doing the following:
 - Making components contextual, with automatic lifecycle management
 - Allowing stateful components to interact through messages

Slide 38

Let us understand the services provided by CDI.

Context and Dependency Injection (CDI) 3-3

Following are the services provided by the CDI in Java EE:

- Provides contexts for the lifecycle of Stateful Session bean.
- Provides dependency injection which allows resources to be injected into the application in a type-safe way.
- Uses expression language for integrating application components.
- Allows associating interceptors with business methods.
- Provides Service Provider Interface (SPI) to integrate third party frameworks into Java EE.
- Decouples client and server through well-defined basic datatypes or object types.
- Decouples the life cycle of the collaborating components.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 38

Use slide 38 to explain the services provided by CDI in enterprise applications.

CDI provides execution environment to the Stateful Session beans through context.

- CDI provides contexts for the lifecycle of Stateful Session bean. Context refers to the environment of creation for the stateful objects.
- CDI provides for dependency injection which allows resources to be injected into the application in a type-safe way.
- CDI allows for integration of different components of the applications by using expression language. Using expression language also enables compatibility among different application components such as JSP, JSF, and so on.
- It allows the bean developer to associate interceptors with bean components in the application. It provides an event-notification model.
- It also provides a Service Provider Interface (SPI) where third party frameworks can be integrated into the Java EE environment.
- CDI provides for loose coupling of the application components:
 - It decouples the client and the server through well-defined basic datatypes or object types and their respective qualifiers.
 - It decouples the lifecycle of the collaborating components, by allowing the stateful components to interact with services by interchanging messages.

Slide 39

Let us understand the process of implementing the managed beans.

Implementing Managed Beans 1-3

All the managed beans are implemented as Java classes.

Following are the requirements of managed bean classes:

- Should not be non-static inner class.
- Should be a concrete class or should be annotated with `@Decorator`.
- Should not be annotated with an EJB component defining annotations.
- Should have a default constructor.
- Should declare a constructor annotated with `@Inject`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 39

Use slide 39 to explain how the managed beans can be implemented as Java classes.

The platform prescribes certain requirements for these classes. Following are the requirements for these classes:

- The managed bean class should not be a non-static inner class.
- The managed bean class should be a concrete class or should be annotated with `@Decorator`.
- The managed bean class should not be annotated with an EJB component defining annotation or declared as an EJB bean class in the deployment descriptor `ejb-jar.xml`.
- The managed bean class should have a default constructor and the class should declare a constructor annotated `@Inject`.

Additional References:

To know more about beans, visit the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/gjebj.html>

Slide 40

Let us understand the process of injecting bean methods through code.

Implementing Managed Beans 2-3




□ Following code snippet demonstrates the usage of `@Inject` annotation:

```

    ...
    import javax.ejb.LocalBean;
    import javax.ejb.Stateless;
    import javax.inject.Inject;
    @Stateless
    @LocalBean
    public class InjectBeanDemo {
        @Inject
        private HelloBean H;
        public InjectBeanDemo(){
            H= new HelloBean();
        }
        public static void main(String args[]){
            InjectBeanDemo I = new InjectBeanDemo();
        }
    }
  
```

A `HelloBean` is injected into the current bean through `@Inject` annotation.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 40

Use slide 40 to explain that the bean object `HelloBean` is injected into the `InjectBeanDemo` class in the private section of declarations through the `@Inject` annotation.

Slide 41

Let us understand the code of the injected bean.

Implementing Managed Beans 3-3




□ Following code snippet shows the `HelloBean` which is injected in `InjectBeanDemo`:

```

package beans;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
@Stateless
@LocalBean
public class HelloBean {

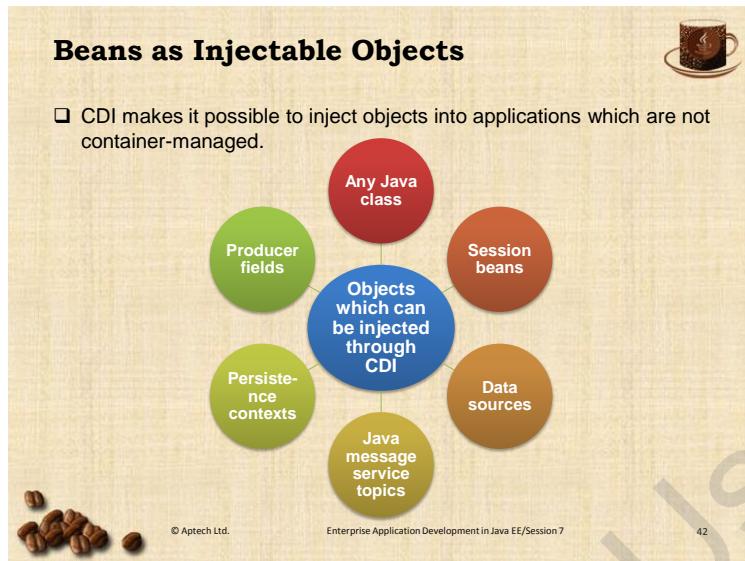
    public HelloBean() {
        System.out.println("In Hello Bean");
    }
}
  
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 41

Use slide 41 to explain the code of the bean which was injected into the `InjectBeanDemo`. This is a simple managed bean which displays a message “In Hello Bean”.

Slide 42

Let us understand the objects which can be injected into an enterprise application.



Use slide 42 to explain the bean objects which are not container managed and those which can be injected into the bean.

- CDI allows any Java class to be injected as a resource into the application.
- CDI enables injection of Session beans as resources.
- CDI enables injection of Java EE resources such as data sources, Java Message Service topics, queues, and so on.
- It allows injection of persistence contexts.
- It allows injection of producer fields. Producer fields are those variables in the class which can be injected into the application.

Additional References:

To get more information on CDI, visit the following link:

<http://jaxenter.com/tutorial-introduction-to-cdi-contexts-and-dependency-injection-for-java-ee-jsr-299-104536.html>

Slide 43

Let us understand how CDI can enable injecting resources into beans.

Injecting Beans

- ❑ Enterprise beans are injected as resources into other bean classes.
- ❑ Enterprise beans are injected using the annotation `javax.inject.Inject`.
- ❑ Following code snippet demonstrates the process of injecting `CustomerInformation` into `LoanApproval` bean:

```
import javax.inject.Inject;
class LoanApproval{
    @Inject CustomerInformation;
    ...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 43

Use slide 43 to explain how enterprise beans are injected as resources into bean classes.

CDI provides `@Inject` annotation to inject a bean into another bean. Bean components can be injected into applications like any other resources.

The code given on slide 43 shows that the `CustomerInformation` bean is injected into `LoanApproval` bean.

Slide 44

Let us understand the process of configuring a CDI application.

Configuring a CDI Application

- ❑ To configure a CDI application, the scope of the bean has to be appropriately defined.
- ❑ The scope of a session bean can have any one of the following values:

Request	Session	Application
Dependent	Conversation	

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 44

Use slide 44 to explain the process of configuring a CDI application. In order to configure the application the scope of the session has to be defined.

The scope of the session can have any one of the following five values:

- **Request** – When the scope of the bean is defined to be a request, then the bean is active only till one request from the client is served.
- **Session** – The scope of the bean in this case is for a set of HTTP requests during an interaction with the client.
- **Application** – When the scope value is set to be application then the bean is active as long as the application is in the container.
- **Dependent** – Dependent is the default scope when no other scope is specified, which implies that the bean object exists to serve exactly one client bean and has the same lifecycle as that of the client bean.
- **Conversation** – This scope is set when there is an interaction with a servlet or similar entities which have long running sessions.

Additional References:

To prepare for a complete example on CDI using Entity bean, visit the following link:

<http://www.mastertheboss.com/jboss-frameworks/cdi/java-ee-6-cdi-example-application>

Slide 45

Let us understand the packaging process of CDI applications.

Packaging CDI Applications 1-2

- ❑ All the beans managed by CDI are packaged as bean archive files.
- ❑ There are two variants of bean archive files:
 - Implicit bean archive files
 - Explicit bean archive files
- ❑ Explicit bean archive comprises `beans.xml` deployment descriptor.
- ❑ Implicit bean archive is one which contains some beans annotated with a scope type.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 45

Use slide 45 to explain the process of packaging the CDI applications.

When you deploy a Java EE application, CDI looks for beans inside bean archive. A bean archive is any module that contains beans that the CDI runtime can manage and inject.

There are two types of archive files into which the bean applications are packaged – implicit bean archive files and explicit bean archive files.

An explicit bean archive comprises a `beans.xml` deployment descriptor. The `beans.xml` file keeps track of all the enterprise beans in the application. This kind of archiving is essential to simplify the bean discovery process in the applications. The `beans.xml` descriptor also defines the interceptors, decorators, and alternatives associated with each bean.

The implicit bean archive is one which contains some beans annotated with a scope type. It does not contain `beans.xml` deployment descriptor. The container checks all the Java classes in the application for bean defining annotation and discovers the beans through them.

Slide 46

Let us understand the code of the deployment descriptor.

Packaging CDI Applications 2-2

- ❑ Following code snippet shows beans.xml deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="
       http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

- ❑ The container checks all the Java classes in the application for bean defining annotation and discovers the beans through them.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 46

Use slide 46 to explain the code of a deployment descriptor beans.xml. The header of the deployment descriptor specifies the standards that are valid in the current version. It defines the XML schema that is used in the current descriptor. It means the CDI can manage and inject any bean in the application.

The bean.xml can be an empty file or can contain the version number 1.1 with the bean-discovery-mode attribute set to the value all.

Tips:

For a Web application, the beans.xml deployment descriptor, if present, must be in the WEB-INF directory. For EJB modules or JAR files, the beans.xml deployment descriptor, if present, must be in the META-INF directory.

In-Class Question:

After you finish explaining the code, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What does the external bean archive contain for CDI?

Answer:

bean.xml file

Slide 47

Let us summarize the session.

Summary

- ❑ Interceptors are invoked when either business methods are invoked or when bean objects are instantiated.
- ❑ Interceptors can be implemented as classes or methods.
- ❑ When interceptors are defined as a class then interceptor method has an `InitialContext` parameter.
- ❑ There can be more than one interceptor defined for the bean class.
- ❑ Multiple interceptors can be defined using `@Interceptors` annotation or through deployment descriptor.
- ❑ CDI is used in enterprise applications for type-safe dependency injection.
- ❑ To configure a CDI application, the scope of the bean has to be appropriately defined for the bean class.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 7 47

Using slide 47, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

7.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concepts of Transactions explained in the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 8 – Transactions

8.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

8.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept of transactions
- Define various properties of transactions
- Explain Java Transaction API (JTA)
- Describe the interfaces of JTA that can be used in the enterprise applications
- Explain programmatic versus declarative demarcation
- Explain container-managed transaction
- Explain bean-managed transaction
- Explain how to manage transactions in messaging

8.1.2 Teaching Skills

To teach this session successfully, you should be aware with the concept of transactions. You should prepare on how to work with Java Transaction API (JTA) to manage the transactions in the enterprise applications.

You should aware yourself with different techniques of transaction management supported by Java EE.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

Objectives

- Explain the concept of transactions
- Define various properties of transactions
- Explain Java Transaction API (JTA)
- Describe the interfaces of JTA that can be used in the enterprise applications
- Explain programmatic versus declarative demarcation
- Explain container-managed transaction
- Explain bean-managed transaction
- Explain how to manage transactions in messaging

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 2

Tell them that they will be introduced to transactions and various aspects of transaction management in enterprise applications. Java Transaction API (JTA) is provided as a part of Java EE and there are various classes and interfaces present in the API and used in applications.

Tell them that they will be introduced to two techniques of transaction management – container managed transactions and bean managed transactions. The session also explains about the message-based transactions.

8.2 In-Class Explanations

Slide 3

Let us understand the basics of transaction.

Transaction Basics

- ❑ Transaction is a group of operations to be executed as a single unit.
- ❑ For example, purchase made through an e-commerce portal involves different operations such as:
 - Choosing the product
 - Adding it to the shopping cart
 - Making payment for the product
 - Finalizing the transaction
- ❑ All these operations are an integral part of an e-commerce transaction.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 3

Use slide 3 to introduce the concept of transactions.

Transaction is a set of operations performed by the application. These operations are expected to be executed as a single unit.

Discuss a scenario of fund transfer performed in a banking application. The fund transfer from an account A to account B involves two operations namely withdrawal or deducting money from account A and depositing money in account B. For the transaction to successfully execute both the comprising operations have to execute successfully or any one of the operation fails, then the whole operation should be terminated.

Discuss the consequences which can occur when the operation is performed successfully on account A, but fails on account B.

Then, discuss the example of transaction explained for an e-commerce portal on slide 3. All the mentioned operations have to successfully execute to complete the transaction.

Then, discuss some of the other domains or business systems where the transaction is employed. For example, Automatic Teller Machine (ATM), Online Book Order, Medical, Flight Booking, and so on.

Slides 4 to 6

Let us understand the relation between transactions and data integrity.

Transaction and Data Integrity 1-3



- An enterprise application stores critical information for business operations in databases.
- If multiple users access the same data at the same time then, the integrity of the data is lost.
- If there is a system failure then, business transaction would partially update the data.
- Thus, transactions ensure data integrity of the data stored in the database.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 8

4

Transaction and Data Integrity 2-3



- Transactions on the database are managed by the transaction management mechanism provided by the DBMS.
- Enterprise applications require transaction management in the middle layer of the application.

The transaction management mechanism in the middle layer is provided by the Java Transaction API (JTA) .



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 8

5

Transaction and Data Integrity 3-3



- Following figure illustrates the concept of transaction using the example of an e-commerce portal:



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 8

6

Use slides 4 to 6 to explain the relation between transactions and data integrity.

Enterprise application comprises real time information of business processes that gets stored in the back-end databases. The data present in the databases should not only be accurate and reliable but also current. If multiple users access the same data at the same time then, the integrity of the data is lost. If any of these transactions fail then it may lead to an inconsistent state of the application database. The transaction management system should ensure that the application does not reach an inconsistent state.

Transaction management system should provide mechanisms to maintain the data integrity in the application.

Use slide 5 to explain the location of the transaction management mechanism in a three-tier application.

The Java transaction management layer lies in the middle layer of the three-tier application. The transactions operate on a database in the EIS layer and are initiated by the user interface tier of the application. The Java Transaction API (JTA) provides classes and interfaces to manage transactions in an application.

Use slide 6 to demonstrate an example of transaction in an e-commerce portal. Provide additional examples such that students can relate to it and understand the concept of transaction in the application.

For instance, when a user clicks the BUY button on an online shopping site, the following processes take place on the server-side:

- User's credit card details are validated.
- Credit card is charged.
- Order is generated and sent to the Shipping department.
- Invoice is sent to the user's e-mail account.

Slide 7

Let us understand transaction scope.

Transaction Scope

- ❑ Transaction scope extends from the begin operation of the transaction and ends at the commit.
- ❑ All operations between the begin and commit have to successfully execute.
- ❑ Transaction is rolled back when any one of the operations in transaction fail.
 - Rollback is the process of restoring the system to previous consistent state.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 7

Use slide 7 to explain the concept of transaction scope.

Scope of transaction starts with the `begin()` method which signifies the beginning of a transaction and ends with either `commit()` or `rollback()` of the transaction.

A transaction can commit only when all the transactions within the transaction scope are executed successfully. When a transaction commits all the changes made by the transaction are made permanent to the application database. When the transaction rolls back, the state of the database is restored to a previous consistent state and the changes made by the operations in the transaction are reversed.

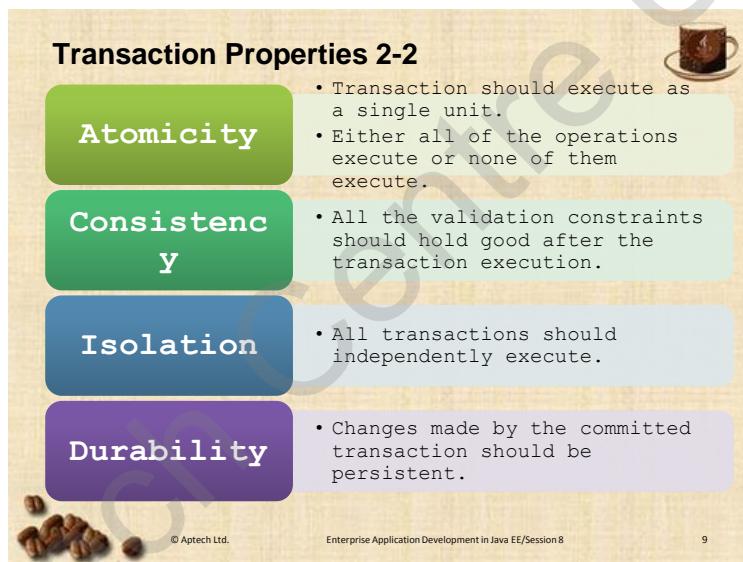
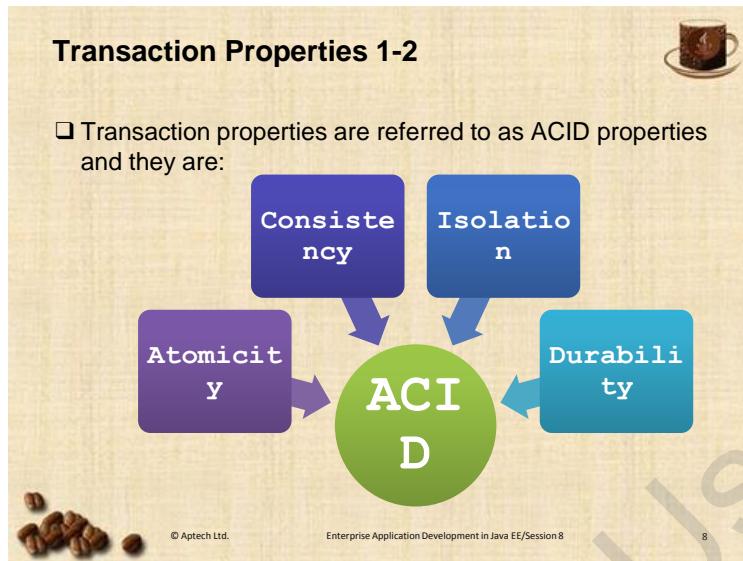
Applications use system log or application log to restore the system to a consistent state before the transaction begins.

For example, the transfer of fund from account A to account B can be step listed as:

```
begin transaction
    debit account A
    credit account B
    update history log
commit transaction
```

Slides 8 and 9

Let us understand the properties a transaction is expected to hold good in an application.



Use slides 8 and 9 to explain the four ACID properties which the transaction has to hold good in the application.

ACID properties refer to the qualities that a transaction should possess.

Atomicity implies that the transaction should execute either completely or should not execute at all. Partial execution of transaction is not acceptable. The transaction should execute as a single unit. If any one of the operations fail then rest of the operations are rolled back resulting in a transaction roll back.

An example of an atomic transaction is an account transfer transaction. In account transfer, the money is removed from account A and placed into account B. If the system fails after removing the money from account A, then the transaction processing system will put the money back into account A, thus bringing the system to its original state.

Consistency of the application data must be retained. After the execution of a transaction the application data should not be left in an inconsistent state. All the constraints of the application should hold good. For instance, after the execution of transaction the age of an employee should not be 500. 500 though a valid numerical value cannot be the age of a person.

In account transfer system, the system is consistent, if the total of all accounts is constant. If an error occurs and the money is removed from account A and not added to account B, then the total in all accounts would have changed. This means that the system would no longer be consistent. Thus, to bring the system in a consistent state, the rolling back is done on the account A to bring the system back in a consistent state.

Isolation of transactions should be ensured. Enterprise applications being multi-user systems can have multiple transactions simultaneously executing. The transaction management system should ensure that the execution of one transaction does not affect other transactions executing in the application.

Durability of transactions is an important requirement. This implies that the changes made by transaction should be made permanent in the application. If the system crashes and recovers then the changes made by all the committed transactions should reflect in the application database.

Additional References:

To get more information about the ACID properties of transaction, refer the following link:

<https://msdn.microsoft.com/en-in/library/aa480356.aspx>

In-Class Question:

After you finish explaining the ACID properties of transaction, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What Atomicity means?

Answer:

Atomicity implies that the transaction should execute either completely or should not execute at all.

Slide 10

Let us understand the concept of transaction demarcation.

Transaction Demarcation

- It is the process of determining where the transaction begins and ends.
- Transaction demarcation can be defined as:
 - Declarative
 - Programmatic
- Declarative demarcation is done through annotations and deployment descriptor.
- Programmatic demarcation is done by the developer by explicitly beginning and ending the transaction.
- Transaction demarcation ends with a commit or rollback operation.
 - The commit request directs all the participating components to store the effects of the transaction operations permanently.
 - The rollback request makes the components to undo the effects of all transaction operations.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 10

Use slide 10 to explain the concept of transaction demarcation in applications.

Transaction demarcation is relevant when developers have to define the beginning and end of the transaction. As a transaction comprises a set of operations, there should be a clear definition of where the transaction begins and where it ends.

There are two methods of transaction demarcation – declarative demarcation and programmatic demarcation.

Declarative transaction demarcation is defined through the deployment descriptor and annotations in the application and these transactions are managed by the container. The container decides when to invoke the begin and commit or abort methods. The container is also responsible to display consistent outcome to all the users.

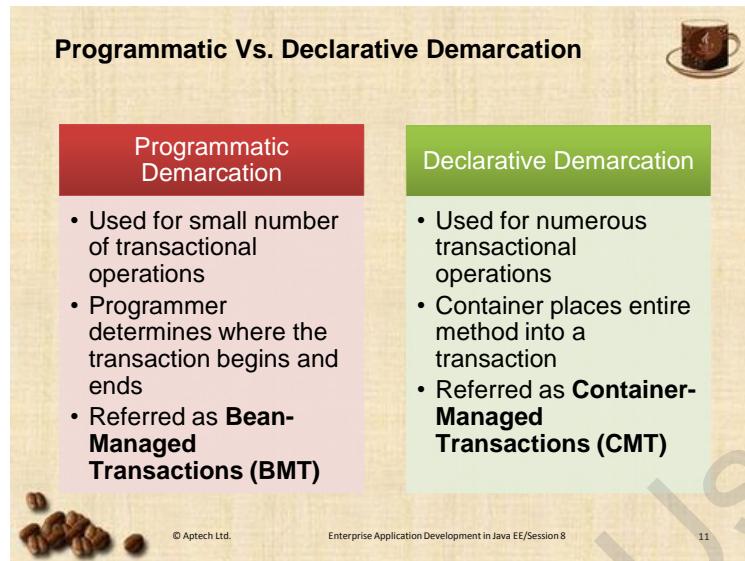
Programmatic transaction demarcation is defined by the developer. The developer will explicitly begin and end transaction through various classes provided by the developer in the code. The developer is responsible for issuing the begin, commit, or abort statement.

Tips:

One of the primary advantages of EJBs is that it allows for declarative transaction management. The transaction can be controlled using the annotations or EJB deployment descriptor.

Slide 11

Let us compare programmatic transaction demarcation with declarative demarcation.



Use slide 11 to compare programmatic transaction demarcation with declarative demarcation.

In programmatic demarcation, developer has to define every aspect of transaction management. However, in case of declarative transaction demarcation, a management policy can be set across all the transactions in the application. Therefore, declarative demarcation is used when the boundaries are to be defined for a large number of transactions. However, in declarative demarcation container is not able to determine the boundaries within the code and places the entire method into transaction.

To define transaction demarcation for small number of transactions programmatic demarcation is used which is also known as bean managed transactions.

Then, tell them that the EJB specifications refer the declarative transactions as Container-Managed Transaction (CMT) and programmatic transactions as Bean-Managed Transaction (BMT).

Slides 12 to 14

Let us understand the components involved in transaction processing.

Transaction Processing 1-3



Following components are involved in transaction processing:

- **Application Components** – EJB business methods which invoke transactions.
- **Resource Managers** – Components which manage the persistent data storage usually drivers.
- **Transaction Managers** – Core component that creates and maintains transactions.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 8 12

Transaction Processing 2-3



When the transaction manager receives a request from the application component to commit a transaction, the transaction manager performs the activity in two phases.

Requests all the involved resource managers to be ready for the transaction, updates all the resources and makes an entry in the transaction log.

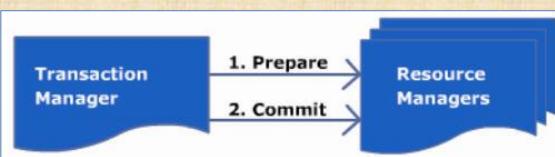
The transaction manager sends a commit request to all the resource managers.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 8 13

Transaction Processing 3-3



Following figure shows the execution of two-phase commit protocol:



 © Aptech Ltd. Enterprise Application Development in Java EE/Session 8 14

Use slides 12 to 14 to explain the transaction processing.

Use slide 13 to explain the application components involved in transaction processing.

There are three main components involved in transaction processing:

- Application components initiate transactions in the application.
- Resource managers inject resources into the components as per requirement of the transaction.
- Transaction managers are responsible for creating, managing, and completing the transactions in the application.

The transaction manager initiates the transaction, carries out all the operations defined in the transaction, and commits the transaction. If some operation in the transaction fails, the transaction manager is responsible for performing the rollback operation.

Use slide 13 to explain the process of committing the transaction using the transaction manager.

When the transaction has to be committed, the transaction manager carries out the operation in two steps.

The first step is that the transaction manager intimates all the resource managers managing the resources used in transactions to update all the resources and make corresponding entries in the transaction log. Once all the updates are done the transaction manager sends a commit request to all the resource managers involved.

Use slide 14 to graphically depict the operations carried out by the transaction manager. This method of committing the transactions is also known as two-phase commit protocol.

In the first phase, all the resource managers are intimated to update the changes made by the transaction and in the second phase the resource managers are requested to commit the changes made.

In-Class Question:

After you finish explaining the components involved in transaction processing, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the role of transaction manager?

Answer:

The transaction manager initiates the transaction, carries out all the operations defined in the transaction, and commits the transaction.

Slide 15

Let us understand Java Transaction API.

Transaction API 1-3

The diagram illustrates the Java Transaction API architecture. It shows a 'Clients use the JTA' box connected to a 'Java Transaction Service' (JTS) component. The JTS is part of an 'EJB Server' which contains 'Container' and 'EJB' components. The EJB Server connects to two 'Database' boxes. A 'commit()' method is shown being called from the client side.

Use slide 15 to explain the implementation of Java Transaction API. There are two important components in Java EE to manage transactions:

- The Java Transaction Service
- Java Transaction API

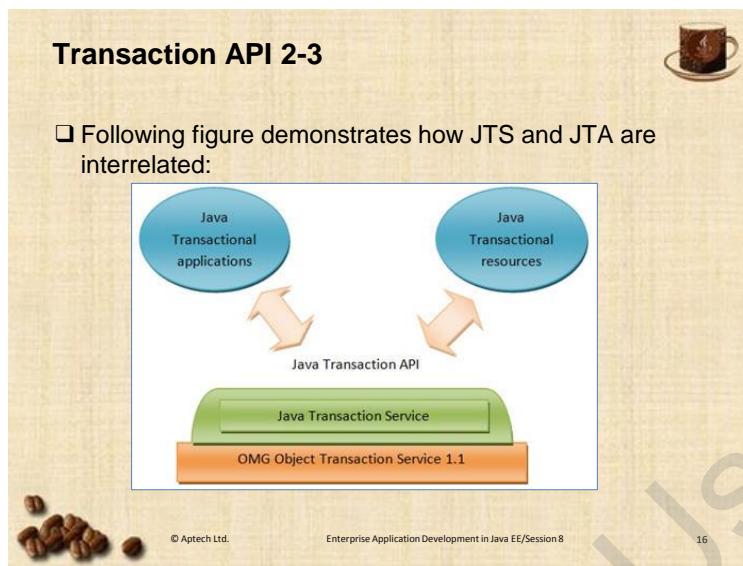
Java Transaction Service implements the specification of transaction manager. Java Transaction API provides interface between the transaction manager and application components.

JTS provides a specification for the implementation of a transaction manager in the application. JTS is based on the Object Transaction Service (OTS) which is a part of the Common Object Request Broker Architecture (CORBA) specification. The JTS service is implemented by the vendors on enterprise middleware to provide transaction processing infrastructure on the application servers.

JTA uses the services and implementation of the transaction manager provided by JTS to implement transaction managing functions.

Slide 16

Let us understand the architecture of Java Transaction API.



Use slide 16 to explain the architecture of JTA.

The transaction manager can interact with the application components through the JTA. The interface provided by the API is also used by the JTS to interact with the resource managers in the application.

Then, explain them JTA API that consists of the following three sets of interfaces:

- The `javax.Transaction.xa.XAResource` interface which is used by JTA to communicate with X/Open XA-enabled resource managers.
- The `javax.transaction.TransactionManager` interface which is used by JTA to communicate with the application servers.
- The `javax.transaction.UserTransaction` interface is used by the enterprise beans and other Java classes to work with the EJB transactions.

Slide 17

Let us understand the different interfaces provided by JTA.

Transaction API 3-3

- ❑ JTA API comprises the following three sets of interfaces:
 - **javax.transaction.xa.XAResource**
 - Is used by JTA to communicate with X/Open XA-enabled resource managers.
 - **javax.transaction.TransactionManager**
 - Is used by JTA to communicate with the application servers.
 - **javax.transaction.UserTransaction**
 - Is used by the enterprise beans and other Java classes to work with the EJB transactions.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 17

Use slide 17 to explain different interfaces provided by the JTA API.

JTA API consists of the following three sets of interfaces:

- The `javax.Transaction.xa.XAResource` interface which is used by JTA to communicate with X/Open XA-enabled resource managers.
- The `javax.transaction.TransactionManager` interface which is used by JTA to communicate with the application servers.
- The `javax.transaction.UserTransaction` interface is used by the enterprise beans and other Java classes to work with the EJB transactions.

In-Class Question:

After you finish explaining the responsibilities of the container, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which interface is used by the enterprise beans and other Java classes to work with the EJB transactions?

Answer:

`javax.transaction.UserTransaction`.

Slide 18

Let us understand container-managed transactions.

Container-Managed Transaction

- ❑ Implements declarative transaction demarcation.
- ❑ Container is responsible for starting and managing the transaction.
- ❑ Container is the demarcation of the transaction.
- ❑ Can work with a Message-driven bean or a session bean.
- ❑ Each transaction is associated with a bean method.
- ❑ Multiple transactions or nested transactions are not allowed in bean methods.
- ❑ Following methods are not used in container-managed transactions:
 - `commit`, `setAutoCommit`, and `rollback` of `java.sql.Connection`.
 - `getUserTransaction()` method of `javax.ejb.EJBContext`.
 - All methods of `javax.transaction.UserTransaction`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 18

Use slide 18 to explain container-managed transactions. In container-managed transactions the transaction scope is defined by the container. The application code does not have an explicit begin and end. The container generally starts a transaction whenever a method execution begins and ends the transaction when the execution of method is complete.

The container is responsible for propagating the context of transaction from one method to another.

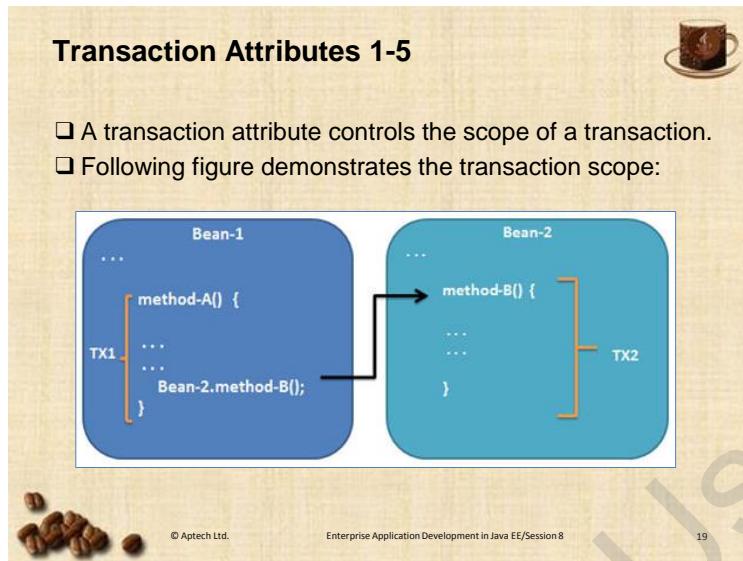
Developers cannot use transaction management statements such as `commit`, `rollback` in the code when the transaction management mechanism is defined to be a container managed transaction.

Following are the methods which cannot be executed in container-managed transactions:

- `commit`, `setAutoCommit`, and `rollback` method of `java.sql.Connection`
- `getUserTransaction` method of `javax.ejb.EJBContext`
- All methods of `javax.transaction.UserTransaction`

Slides 19 and 20

Let us understand transaction attributes.



Use slide 19 to explain transaction scope. In case of container-managed transactions the scope of a transaction starts when a method begins and ends when a method ends. A new transaction begins when a new method is called and the scope ends when the method returns. However, all methods in an application do not require a transaction scope. This is explicitly specified through transaction attributes.

Discuss with them a scenario where:

Method-A begins a transaction and then, invokes method-B of Bean- 2. When method-B executes, a point which can be asked is, whether method-B runs within the scope of the transaction started by method-A? or does it execute with a new transaction?

The answer to these questions, depends on the scope of transaction attribute selected by method-B.



Use slide 20 to explain transaction attributes. The container initiates a transaction for every method executed in the application. Some methods do not require a transaction context while executing the method. Whether a method requires a transaction context while execution or not has to be specified as transaction attribute for the method associated with the transaction.

There are six different transaction attributes – **Required**, **RequiresNew**, **Mandatory**, **Never**, **Supports**, **Not Supported**. Each of these attributes conveys information about the transaction context.

- When the transaction attribute value is set to **Required**, then it implies that the bean method should be invoked within a transaction scope.
- The transaction attribute **RequiresNew** implies that the execution of the annotated bean method requires a new transaction scope to execute. The new method cannot execute from the transaction scope from which it is invoked.
- When the transaction attribute is set to **Mandatory**, it implies that a transaction scope is required in order to execute the annotated method.
- When the transaction attribute value is set to **Never**, then it implies that the annotated bean method should not be invoked in a transaction scope.
- When the transaction attribute value is set to **Supports**, it implies that when an annotated bean method is invoked from a transaction scope then, the invoked bean method executes in the current transaction scope.
- When the transaction attribute value is set to **Not Supported**, it implies that the annotated method cannot be executed in a transaction scope.

Slide 21

Let us understand transaction attributes and their usage with different beans.

Transaction Attributes 3-5



Following table shows the list of transaction attributes permitted according to the EJB type:

Transaction Attribute	Stateless Session Bean	Stateful Session Bean	Entity Bean	Message-Driven Bean
Not Supported	Yes	Yes	Yes	Yes
Required	Yes	Yes	Yes	No
Supports	Yes	Yes	Yes	No
RequiresNew	Yes	No	No	No
Mandatory	Yes	No	No	Yes
Never	Yes	No	No	No



© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 21

Use slide 21 to explain which transaction attribute can be applied to which Session beans. The table shows that all the transaction attributes can be applied on Stateless Session beans. **RequiresNew**, **Mandatory** and **Never** cannot be applied on Stateful Session beans and Entity beans. **Required**, **Supports**, **RequiresNew** and **Never** cannot be applied on Message-driven beans.

Tips:

Current version of Java EE does not use entity beans widely. Entity beans are replaced by POJO classes which are referred as entities in JPA.

Additional References:

To get more information on transaction attributes, refer the following link:
<http://www.javabeat.net/managing-transactions-in-ejb-3-0/>.

In-Class Question:

After you finish explaining the transaction attributes, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which transaction attributes are supported by Stateless, Stateful, and Entity Bean, but not Message-driven bean?

Answer:

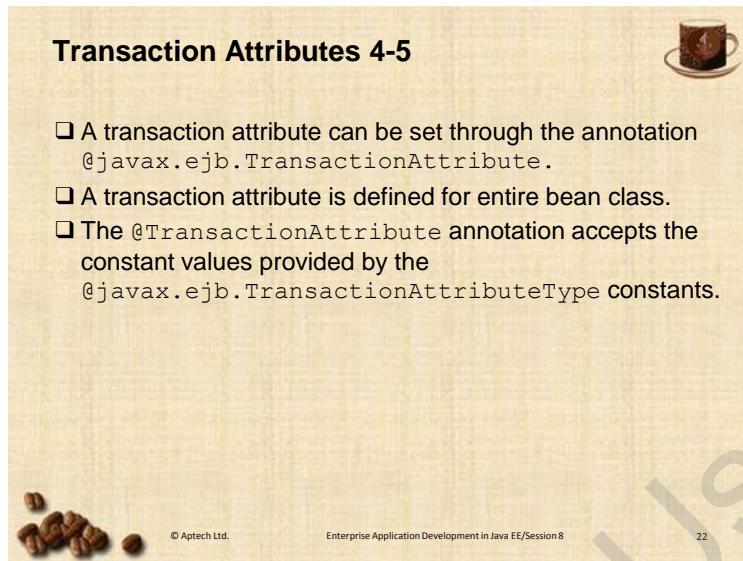
Requires and Supports.

Slides 22 and 23

Let us understand how to set transaction attributes in the application code.

Transaction Attributes 4-5

- A transaction attribute can be set through the annotation `@javax.ejb.TransactionAttribute`.
- A transaction attribute is defined for entire bean class.
- The `@TransactionAttribute` annotation accepts the constant values provided by the `@javax.ejb.TransactionAttributeType` constants.

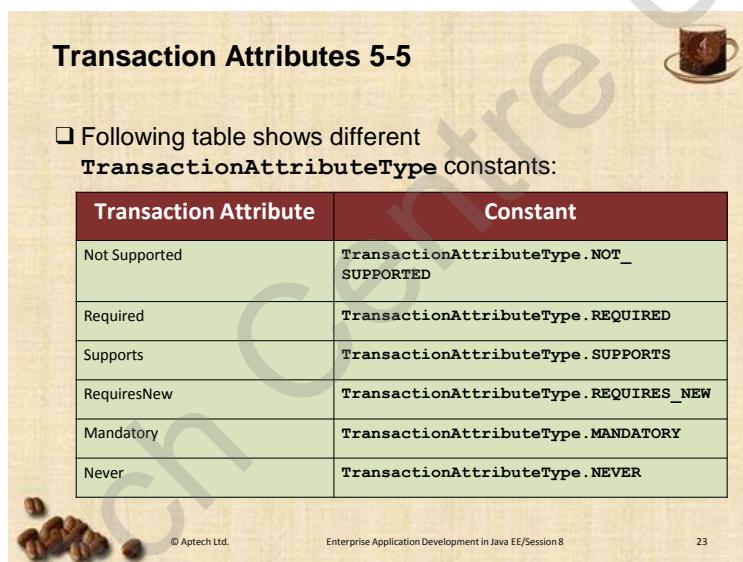


© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 22

Transaction Attributes 5-5

- Following table shows different `TransactionAttributeType` constants:

Transaction Attribute	Constant
Not Supported	<code>TransactionAttributeType.NOT_SUPPORTED</code>
Required	<code>TransactionAttributeType.REQUIRED</code>
Supports	<code>TransactionAttributeType.SUPPORTS</code>
RequiresNew	<code>TransactionAttributeType.REQUIRES_NEW</code>
Mandatory	<code>TransactionAttributeType.MANDATORY</code>
Never	<code>TransactionAttributeType.NEVER</code>



© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 23

Use slides 22 and 23 to explain the annotations that can be used in the application code to set transaction attributes.

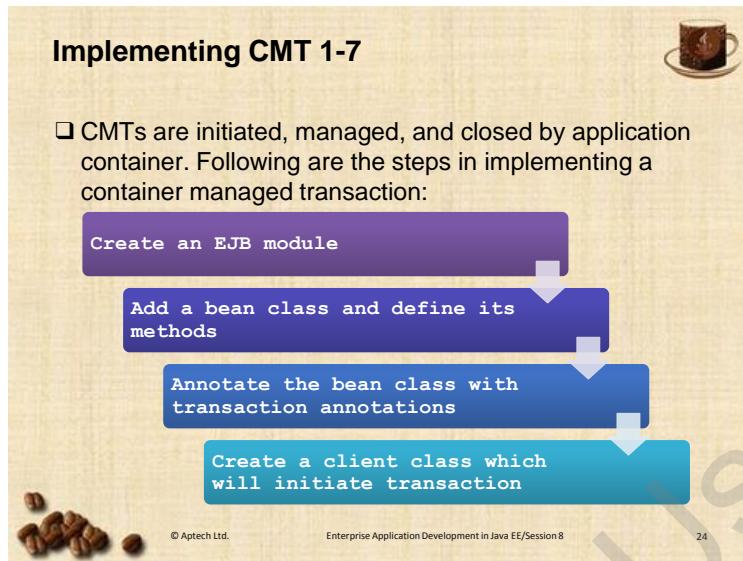
The transaction attribute for a method can be defined by prefixing the method with `@javax.ejb.TransactionAttribute`.

When the attribute is prefixed for entire bean class, then it is applied to all the methods of the bean class. The transaction attribute type can assume one value among the six types. The transaction attribute type can be set through `@javax.ejb.TransactionAttributeType`.

Use slide 23 to explain annotations used to set the transaction attribute type. Each of the transaction attribute can be set to one of the six values with the annotations as shown on the slide.

Slide 24

Let us understand the process of creating container-managed transactions in an EJB module.



Use slide 24 to demonstrate the steps to be followed to implement a container managed transaction in the application.

Following are the steps involved in implementing the transaction:

1. Create an EJB module.
2. Add a bean class and define methods in the class.
3. Annotate the bean methods appropriately according to the application requirement. In this case, perform a transaction on the database where a row is inserted into the database table.
4. Create a client class which will initiate the transaction.

Slides 25 to 27

Let us understand the code which implements a container-managed transaction.

Implementing CMT 2-7



Following code snippet demonstrates a container managed transaction:

```

...
@TransactionManagement(TransactionManagementType.CONTAINER)
@Stateless
@LocalBean
public class CMT {
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public String add(Connection c){
        String message;
        try{
            c.createStatement();
            CachedRowSet st = new CachedRowSetImpl();
            String query = "insert into Customers
                           values(1007,'Martha')";

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 25

Implementing CMT 3-7



```

message="Row Inserted";
}catch(SQLException e){ message=e.toString(); }
return message;
}

@TransactionAttribute(TransactionAttributeType.REQUIRED)
public CachedRowSet display(Connection c) throws
    SQLException{
    Statement stmt = c.createStatement();
    CachedRowSet st = new CachedRowSetImpl();
    String quer = "select * from Customers";
    st.setCommand(quer);
    st.execute(c);
    return st;
}
}

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 26

Implementing CMT 4-7



- The code shows a bean class with two bean methods defined.
- The annotation in the code defines the transaction management type as Container.
- This implies that the transaction management is taken care of by the application container.
- Annotations define the transaction attributes for each of the bean methods.
- The bean method `add()` takes a `Connection` object as a parameter.
- The bean method `display()` also connects to the database through a `Connection` object and displays all the rows present in the `Customers` table.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 27

Use slides 25 to 27 to explain the code for implementing container-managed transactions.

The annotation `@TransactionManagement` specifies that the type of transaction management used in the application is container based.

The `@TransactionAttributeType` annotation is used to define the attribute as REQUIRED. This implies that a transaction context is required to execute this method.

There are two methods both of which require a transaction context according to the definition, which means that the container has to create a transaction context while executing these methods.

Explain them that the code snippet shows a bean class with two bean methods defined. The annotation in the code defines the transaction management type as Container. This implies that the transaction management is taken care of by the application container. Annotations define the transaction attributes for each of the bean methods.

The bean method `add()` takes a `Connection` object as a parameter. This object is used to connect the bean to the database of the application. The method then initiates a transaction on the database to add a row to the table.

The bean method `display()` also connects to the database through a `Connection` object and displays all the rows present in the `Customers` table.

Slide 28

Let us understand the client code which initiates these transactions.

Implementing CMT 5-7

- Following code snippet shows the client code which accesses bean methods:

```

    ...
    public class CallCMT {
    public static void main(String[] args)
    try {
        String driver = "org.apache.derby.jdbc.ClientDriver";
        String url = "jdbc:derby://localhost:1527/sample";
        String username = "app";
        String password = "app";
        Class.forName(driver).newInstance();
        Connection conn = DriverManager.getConnection(url,
                                                      username, password);
        CMT c = new CMT();
        c.add(conn);
        c.display(conn);
    }catch(){ . . . }
    }
  
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 28

Use slide 28 to explain the client code which invokes the transaction in the application. The client is establishing a connection with the **sample** database. The **sample** database is located on the server which can be accessed through the credentials provided.

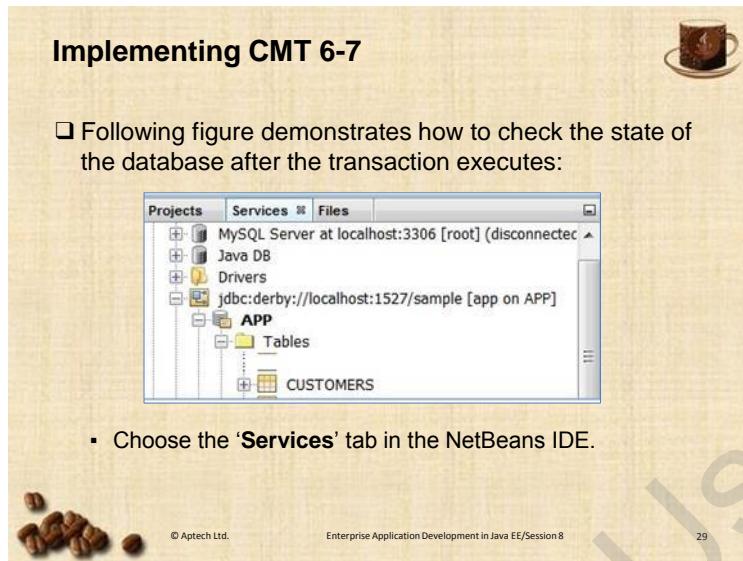
After the connection establishing step is successful, methods defined in the CMT class are invoked. Here, the client is invoking two different transactions.

Tips:

In order to execute the client code and corresponding transactions, the developer has to define a database in the EJB module. The database should be according to the JDBC url as given on slide 28. If the database url is something else, then the code has to be modified accordingly.

Slide 29

Let us understand the database defined through NetBeans IDE.



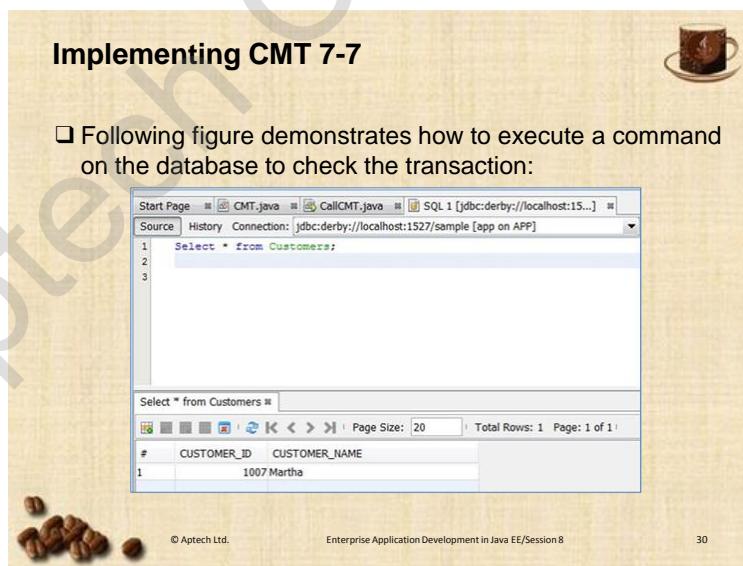
Use slide 29 to explain the location of the database defined in NetBeans IDE. You can locate the database in the 'Services' tab.

Tips:

Developers can check the result of the transaction on the database by executing SQL commands on the database.

Slide 30

Let us execute an SQL command on the database.



Use slide 30 to explain the execution of SQL command on the database. It would be preferable if you can demonstrate execution of the command through NetBeans IDE.

Slide 31

Let us understand how transactions are rolled back.

Rollback Transaction

❑ A transaction committed by CMT can be rolled back in the following ways:

- If a system exception is raised, then the container automatically rolls back the transaction.
- If an application exception is raised, then the container will not rollback it automatically. The EJB must invoke the method `setRollbackOnly()` of the `EJBContext` interface to notify the container to roll back the transaction.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 31

Use slide 31 to explain the process of rolling back a transaction. A transaction has to be rolled back, if any of the operations in the transaction fail or an exception has occurred in any of the operations.

If an application exception is raised, then the container will not rollback the transaction automatically. The EJB must invoke the method `setRollbackOnly()` of the `EJBContext` interface to notify the container to rollback the transaction.

Slides 32 and 33

Let us understand the developer's responsibilities when the transactions are to be container managed.

Developer Responsibilities 1-2



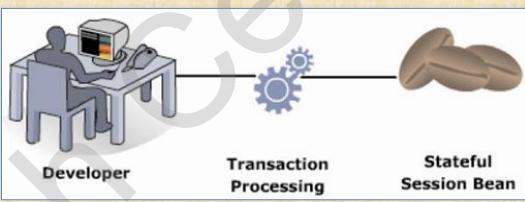
- The developer has to set the transaction attributes in the deployment descriptor.
- The developer should handle the exceptions appropriately.
- The developer should implement `javax.ejb.SessionSynchronization` interface in case of Stateful Session bean.
- The methods used for handling transactions in Stateful Session beans are:
 - `afterBegin()`
 - `beforeCompletion()`
 - `afterCompletion()`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 32

Developer Responsibilities 2-2



- Following figure shows the developer's responsibilities with respect to Stateful Session beans:



The diagram illustrates the developer's responsibilities in the context of Stateful Session beans. It features a developer icon sitting at a desk with a computer monitor, connected by a line to a gear icon labeled "Transaction Processing", which is further connected to a coffee bean icon labeled "Stateful Session Bean".

- During the creation of a Stateful Session bean, a developer can implement `javax.ejb.SessionSynchronization` interface, which provides methods for persisting the state at various stages.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 33

Use slides 32 and 33 to explain the responsibilities of the developer. When the transaction management technique used in the application is container managed transaction then the developer has to provide certain parameters to the container. The container configures its transaction management according to these parameters.

The developer should provide appropriate transaction attributes in the deployment descriptor. If the developer anticipates exceptions in the application then the developer should provide appropriate exception handling for such exceptions.

In case of Stateful Session beans the developers should implement the interface `javax.ejb.SessionSynchronization`. This interface has methods `afterBegin()`, `beforeCompletion()`, and `afterCompletion()` which have to be defined by the developer while implementing the interface.

The purpose of implementing these methods is to persist the state of the Entity represented by the Stateful Session bean.

afterBegin()

The `afterBegin()` method notifies the EJB instance that a new transaction has started. At this point, the method can save state information about the bean prior to the transaction commitment. The EJB instance can use this method to read data from the database and cache the data in the instance fields.

Syntax:

```
public void afterBegin() throws EJBException, java.rmi.  
RemoteException
```

where,

`EJBException`: is raised to indicate a failure caused by a system-level error.

`java.rmi.RemoteException`: is raised to indicate communication-related exceptions that occur during the execution of a remote method call.

beforeCompletion()

The `beforeCompletion()` method notifies the EJB instance that a transaction is about to complete. The Session bean can use this method to rollback the transaction, if required. The Session bean can also use the `beforeCompletion()` method to update the database with the values of the instance variables.

Syntax:

```
public void beforeCompletion() throws EJBException, java.  
rmi.RemoteException
```

afterCompletion(boolean committed)

The `afterCompletion()` method indicates that the container tried to commit the transaction, and whether it succeeded or failed. If a rollback occurred, the Session bean can refresh its instance variables from the database in the `afterCompletion()` method. This method has a single `boolean` parameter whose value is true if the transaction was committed and false if it was rolled back.

Syntax:

```
public void afterCompletion(boolean committed) throws EJBException,  
java.rmi.RemoteException
```

Slide 34

Let us understand what a synchronized Stateful Session bean is.

Synchronized Stateful Session Bean

- Following code snippet shows an example of a synchronized Stateful Session bean:

```

    ...
    @Stateful
    public class MyStatefulEJB implements
    SessionSynchronization {
    protected int total = 0; // actual state of the bean
    // value inside transaction, not yet committed
    protected int newtotal = 0;
    protected String clientUser = null;
    protected javax.ejb.SessionContext sessionContext = null;
    public void ejbCreate(String user)
    {
        total = 0;
        newtotal = total;
        clientUser = user;
    }

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 34

Use slide 34 to explain a synchronized Stateful Session bean.

Following is the code of methods of synchronized Stateful Session bean which are implemented in the Stateful Session bean:

```

//Callback methods

//Called just after the start of a transaction
public void afterBegin() { newtotal = total; }

//Called just before the end of a transaction
public void beforeCompletion() { total = newtotal; }

//Called just after the end of a transaction
public void afterCompletion(boolean committed) { if (committed) {
total = newtotal;
}
else { newtotal = total; }
}

public void buy(int s)
{ newtotal = newtotal + s; return; }

public int red() { return newtotal; }
}

```

Slide 35

Let us understand the container responsibilities.

Container Responsibilities

- ❑ Based on the transaction attribute, the container can perform one of the following actions before performing the transaction:
 - Continue the current transaction.
 - Suspend the current transaction and run the method without a transaction.
 - Suspend the current transaction and begin a new one.
 - Refuse to execute the method at all.
- ❑ At the end of a method call, the container will attempt to commit or rollback any transaction.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 35

Use slide 35 to explain the tasks which are to be carried out by the container provided the developer provides required parameters.

The container associates a transaction with a method. The transaction attributes are set through the deployment descriptor in the application. When the container has to start a new transaction, that is, invoke a new method it has to perform one of the following four steps:

- Continue the current transaction
- Suspend the current transaction and run the method without a transaction
- Suspend the current transaction and begin a new one
- Refuse to execute the method at all

The container has to either commit or rollback the changes made by the transaction at the end of the method.

In-Class Question:

After you finish explaining the responsibilities of the container, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Definition of which of the methods cannot be present in the code of an application which uses container-managed transactions?

Answer:

Methods of the class UserTransaction, commit(), rollback(), and begin() should not be defined in the application if the application is using container-managed transactions, as these methods are used for managing transactions.

Slide 36

Let us understand bean-managed transactions.

Bean-Managed Transactions 1-2

- ❑ Bean-managed transactions are useful when a method has to associate with more than one transaction.
- ❑ Bean-managed transactions provide better control on the application execution.
- ❑ Developer is responsible for starting the transaction and then committing or rolling back the transactions to end it.
- ❑ Bean-managed transactions are implemented through Java Database Connectivity (JDBC) or JTA.
- ❑ Bean-managed transaction is an instance of **UserTransaction**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 36

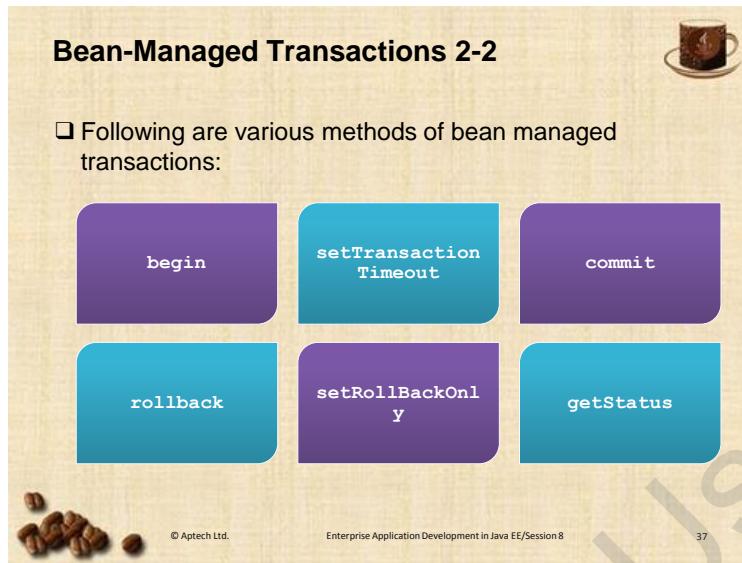
Use slide 36 to explain bean-managed transactions. In case of bean-managed transactions developers should write code to define the transaction boundaries and execute it. A bean-managed transaction is implemented through the `UserTransaction` class provided by JTA. Earlier versions of Java EE used Java Database Connectivity (JDBC) to implement transactions.

In order to use a bean-managed transaction demarcation, an instance of the class `UserTransaction` is to be obtained. This object can be obtained through `getUserTransaction()` method. A `UserTransaction` object enables the developer to define the transaction boundaries.

When the transaction demarcation is programmed into the beans, the transactional behavior cannot be changed. However, this is possible in declarative transaction demarcation.

Slide 37

Let us understand the methods of UserTransaction class for bean-managed transactions.



Use slide 37 to explain the methods provided by the `UserTransaction` class for bean-managed transactions.

- **`begin`** – begin method is used to start a transaction context.
- **`setTransactionTimeout`** – Transaction managers have to set a time limit on the transactions that are executing. The transaction must rollback if it does not complete before the timeout period expires. The timeout value is an integer measuring the time in seconds.
- **`commit`** – commit method is used to commit a transaction. If the method associated with the committed transaction fails for some reason, it will result in a **`RollBackException`**.
- **`rollback`** – this method is used to rollback the associated transaction.
- **`setRollBackOnly`** – this modifies the method such that the transaction is finally rolled back.
- **`getStatus`** - this method returns the status of the transaction associated with the method.

Slides 38 and 39

Let us understand bean-managed transactions.

Implementing BMT 1-2



Following code snippet shows a servlet code which explicitly initiates a transaction:

```
public class BMTServlet extends HttpServlet {
    @EJB
    Private BMT b;
    @Resource
    javax.transaction.UserTransaction utx;
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try{
            /* TODO output your page here. You may use following sample code.
            */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 38



Implementing BMT 2-2



```
out.println("<title>Servlet BMTServlet</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>CUSTOMER DETAILS</h1>");
String driver = "org.apache.derby.jdbc.ClientDriver";
String url = "jdbc:derby://localhost:1527/sample";
String username = "app";
String password = "app";
Class.forName(driver).newInstance();
Connection conn = DriverManager.getConnection(url, username, password);
System.out.println("Connection Done");
utx.begin();
out.print(BMT.add(conn));
...
utx.commit();
out.println("</body>");
out.println("</html>");

}
catch(Exception ex){
    out.println("Error: Other exception - " +ex);
}
}
.
.
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 39

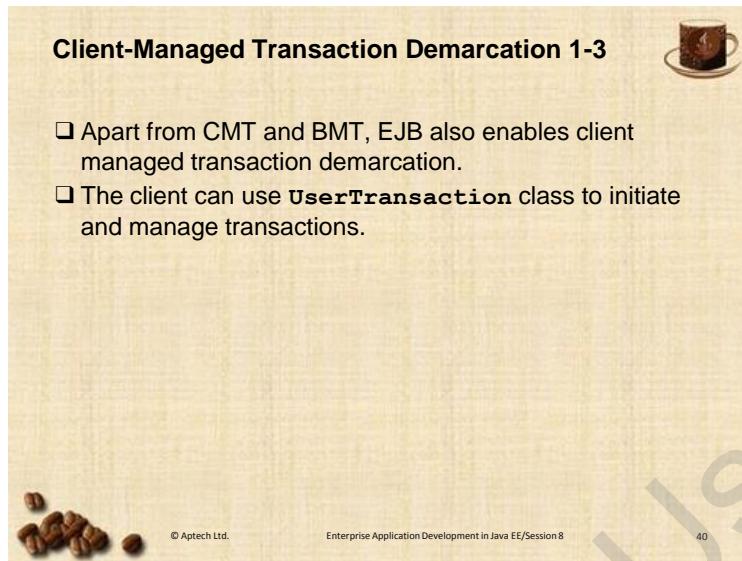


Use slides 38 and 39 to explain bean-managed transactions.

In the given code a transaction is initiated through a servlet in the application. The servlet establishes a connection with the database by providing appropriate credentials. The developer begins a user transaction after establishing connection with the database. The transaction is started with `utx.begin()` statement. A Stateful Session bean is invoked and then the transaction is committed. An exception handling block is defined to handle exceptions which might arise in the transaction.

Slide 40

Let us understand client-managed transaction demarcation.



The slide has a light beige background with a subtle coffee cup and beans graphic. At the top left, the title "Client-Managed Transaction Demarcation 1-3" is displayed in a bold, black, sans-serif font. In the top right corner, there is a small icon of a coffee cup on a saucer. Along the bottom edge, there are three small decorative elements: a cluster of coffee beans on the left, the copyright notice "© Aptech Ltd." in the center, and the slide number "40" on the right.

- ❑ Apart from CMT and BMT, EJB also enables client managed transaction demarcation.
- ❑ The client can use **UserTransaction** class to initiate and manage transactions.

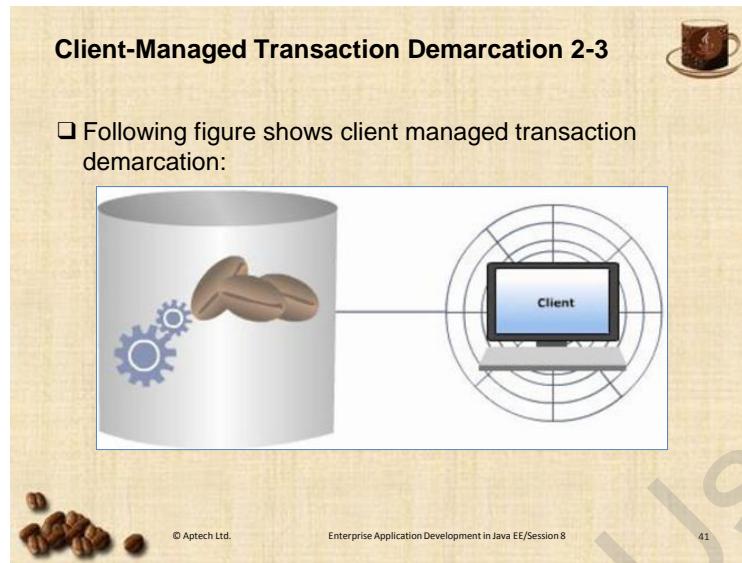
Use slide 40 to explain client-managed transaction demarcation. Java EE provides **UserTransaction** class to enable applications clients to start transactions.

The clients may be JSPs, Servlets, or stand-alone clients. The process of implementing client-managed transaction demarcation is similar to BMT.

Clients can initiate transactions through JNDI lookup. The client gets a `javax.transaction.UserTransaction` object and calls the `begin()`, `commit()`, and `rollback()` methods as required.

Slide 41

Let us understand a client initiated transaction.



Use slide 41 to explain how application clients can initiate transactions in the application. The image shows a Web client initiating a transaction on the server.

Slide 42

Let us understand how the clients perform JNDI lookup.



Use slide 42 to explain the code through which clients can initiate transactions through JNDI lookup.

Slide 43

Let us understand how to apply transactions to messaging.

Applying Transaction to Messaging 1-2

- ❑ Transactions may also include sending and receiving messages.
- ❑ Java application uses Message-driven beans and JMS messages to send and receive messages.
- ❑ Message-driven beans can work with both CMT and BMT.
- ❑ Message-driven beans support only two transaction attributes – **Required** and **NotSupported**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 43

Use slide 43 to explain how messages and transactions are related. A transaction may have to send and receive messages also.

Consider a scenario where an individual performs a cash withdrawal at an ATM. The application sends a message to the user's mobile and an e-mail informing about the transaction.

An ATM transaction includes the following operations:

- User logs in to the account using valid credentials
- Enters the amount to be withdrawn
- System performs a check in the account whether there is sufficient balance or not
- If there is sufficient balance the application directs the ATM to dispense the cash
- System sends a message to the user

In order to receive and process messages, Java application uses message-driven beans. These message-driven beans can work with both container-managed and bean-managed transactions.

The processing of the message received is defined in the `onMessage()` method of the message-driven bean.

Slide 44

Let us understand the association of transactions with message-driven beans.

Applying Transaction to Messaging 2-2

Following figure shows using transactions with Message-driven beans:

Message-driven Beans

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 44

Use slide 44 to demonstrate the relation between transaction and message-driven beans.

Slides 45 and 46

Let us understand transaction isolation.

Database Locking and Isolation 1-2

- ❑ When multiple transactions are simultaneously executing, they are scheduled on the database.
- ❑ To ensure consistency, the database provider should ensure transaction serializability.
- ❑ Locking mechanism is used to provide transaction serializability.
- ❑ There are two types of locks which can be acquired by the transaction:
 - Shared or read lock
 - Exclusive or write lock

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 45

Database Locking and Isolation 2-2

Following figure shows different types of locks:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 46

Use slides 45 and 46 to explain transaction isolation. Isolation of transactions is an essential feature which has to be implemented. When multiple transactions are simultaneously executing, they should not affect each other. Each transaction should execute independent of each other and the final resultant database should be equivalent to what it would be when the transactions are executed in a serial order.

Java EE provides locking mechanisms to achieve transaction isolation in concurrent transactions.

There are two types of locks which the application can acquire on data being accessed – shared lock and exclusive lock.

Shared lock is acquired by an application to read the data, it enables other transactions also to read the data locked. It is hence called a Read lock, transactions intending to modify this data have to wait until the lock is released.

Exclusive lock is acquired by a transaction which intends to perform write operation. No other transactions can either read or write the locked data.

In-Class Question:

After you finish explaining transaction isolation and locking mechanisms, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which lock should a transaction acquire when the client has initiated a transaction to know the balance of an account?

Answer:

The transaction must acquire a read lock as the transaction would not need to modify the balance in the account.

Slide 47

Let us summarize the session.

Summary

- ❑ Transactions are a set of operations which has to execute as a single unit.
- ❑ Transactions should have properties such as atomicity, consistency, isolation, and durability.
- ❑ Transactions are supported in enterprise applications through Java Transaction Services and Java Transaction API.
- ❑ Transactions can be managed both declaratively and programmatically.
- ❑ Declarative transaction management makes use of annotations, deployment descriptors, and transaction attributes. They are also known as container-managed transactions.
- ❑ Explicit transaction management makes use of the JTA and its interfaces for transaction management, such transactions are also known as bean-managed transactions.
- ❑ Transactions can execute along with Stateful, Stateless, and Message-driven beans.
- ❑ Messages in transactions can be handled through both container-managed transactions and bean-managed transactions.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 8 47

Using slide 47, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

8.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concept of Entity Persistence in Java enterprise application that is offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 9 – Persistence of Entities

9.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

9.1.1 Objectives

By the end of this session, the learners will be able to:

- Understand how to use JDBC API for persisting data
- Explain Object Relational Mapping (ORM)
- Describe the ORM tools used for performing data persistence
- Explain Java Persistence API (JPA)
- Describe Entities and Entity Manager in JPA
- Understand how to manage entities using JPA
- Understand how persistent objects are mapped onto the database

9.1.2 Teaching Skills

To teach this session successfully, you should be aware of what databases are and how they are used in enterprise applications. You should be aware of how Java applications use Java Database Connectivity (JDBC) and Java Persistence API (JPA) to connect to a database and perform operations on it. You should also be aware of how the data conversion happens between Java objects and relational tables through object relational mapping techniques.

The session explains the classes in JPA used for managing and retrieving data from the database. It also explains how persistent objects are mapped onto the application database.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- Understand how to use JDBC API for persisting data
- Explain Object Relational Mapping (ORM)
- Describe the ORM tools used for performing data persistence
- Explain Java Persistence API (JPA)
- Describe Entities and Entity Manager in JPA
- Understand how to manage entities using JPA
- Understand how persistent objects are mapped onto the database

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 2

Using slide 2, tell the students that they will be introduced to JDBC API and JPA in this session. JDBC is an API used by earlier versions of Java to connect to the database. The JDBC API is a low-level API used by Java developers to store and retrieve data from the relational database through Structured Query Language (SQL). Developers write SQL statements to perform operations on the relational database. Tell the students that JPA and ORM are explained in this session.

9.2 In-Class Explanations

Slide 3

Let us understand relational databases and their association with Java applications.

Introduction

- ❑ Enterprise applications persist data on relational databases.
- ❑ **Relational databases**
 - Stores data in the form of tables.
 - Each table in a relational database comprises rows and columns.
- ❑ **Java SE platform**
 - Introduced **JDBC API** for persistence of data in the relational database.
 - **JDBC API** is a low-level API used by Java developers to store and retrieve data from the relational database through Structured Query Language (SQL).
 - Allows the developers of Java applications to establish a connection with the database using a driver.
- ❑ Entity Beans were introduced by Java EE for persistence of objects.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 3

Use slide 3 to explain the concept of relational databases and their utility in enterprise applications. Enterprise applications have to process huge volumes of application data. Relational databases are based on relational model where the data is organised in the form of tables. Each table has data arranged in rows and columns.

Java being an object-oriented programming language manages the application data as objects. Each object is associated with data and behaviour which is represented as variables and methods of the class. Therefore, there should be some mechanism which bridges the gap between the object-oriented interpretation of data and relational interpretation of data.

JDBC API of Java EE platform is used to bridge the gap between the databases and object-oriented applications. JDBC allows developers to execute SQL queries on the database and fetch the results into the Java Application. JDBC requires database drivers to be installed and configured in order to connect the Java Application with the relational database.

Java EE provided Entity beans for all the database related operations. Therefore, data stored in the database can be managed easily through the Entity beans.

Slide 4

Let us understand the components of JDBC API.

Persistence Using JDBC API 1-2

- ❑ It is a set of classes and interfaces to programmatic access the database – `java.sql` and `javax.sql`.
- ❑ Following are the components of the JDBC API:

JDBC API		
JDBC Driver Manager	JDBC Test Suite	JDBC-ODBC Bridge

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 4

Use slide 4 to explain the components provided by the JDBC API to manipulate databases. The API provides two classes which can be used by the developer – `java.sql` and `javax.sql`

The two packages defined for JDBC API are as follows:

java.sql - This package contains the API for storing and retrieving data from the relational database.

javax.sql - This package contains the API for processing data from the relational database in the Java applications running on the middle-tier servers. These classes have advanced methods and sub classes which enable the developer to write code to connect the application with the database, to execute SQL queries on the database and retrieve results after the query execution.

Apart from these classes, JDBC API provides three components – JDBC Driver Manager, JDBC Test Suite and JDBC-ODBC Bridge.

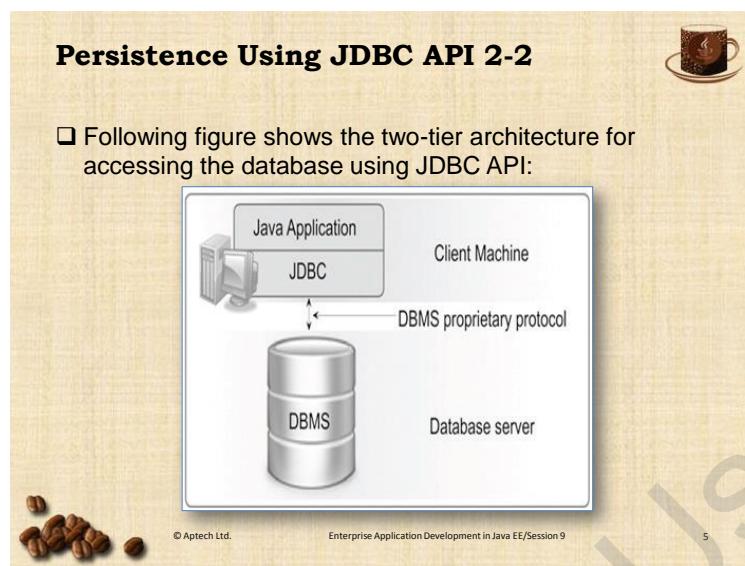
The JDBC Driver Manager is a concrete Java class which keeps track of the database drivers available. The availability of the driver for a specific database implies that the Java application can be connected with that database.

JDBC Test Suite is used to check whether the JDBC drivers will run the application developed by the developer.

The JDBC-ODBC is a Java software bridge driver that connects the JDBC API to the relational database through ODBC drivers. The ODBC binary code needs to be loaded on every machine using JDBC-ODBC bridge driver. Using these drivers the application can connect to almost any RDBMS database system that supports ODBC drivers. The ODBC acts as middle tier in the 3-tier applications. It removes the overhead of re-writing code to connect to every RDBMS system. Same code can be used to connect to the RDBMS systems with little modifications.

Slide 5

Let us understand the two-tier architecture of the database access in applications.



Use slide 5 to explain the two-tier architecture of database access in applications. The Java application accesses the JDBC API and connects to the JDBC driver provided by the application. This comprises the first tier of the application.

The second tier of the application comprises the actual database which is implemented through a relational database.

Slides 6 to 8

Let us understand how to implement the JDBC APIs.

Implementing JDBC API 1-3



Following are the steps to process SQL statements using JDBC API:

- Loading the Drivers**
 - The JDBC driver provides connection to the database and helps in transferring queries and their results between Java application and relational databases.
- Establishing connection**
 - Driver should be loaded and initialized.
 - DriverManager or DataSource class are used to establish connections.
 - Connection object implies a database connection.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 6

Implementing JDBC API 2-3



Creating SQL Statements

- Created using interfaces such as Statement, PreparedStatement, and CallableStatement.
- createStatement() method of Connection class is used to create an SQL statement.

Executing the Query

- Statements are executed using methods execute(), executeQuery(), and executeUpdate().
- ResultSet object is returned when queries are executed.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 7

Implementing JDBC API 3-3

Processing ResultSet object

- The ResultSet object holds the data retrieved from the query executed in the database.
- The obtained ResultSet object is further processed in the application using a cursor.

Closing the Connection

- close() method of the Statement interface can be used to close the statement on completion.
- close() method frees the allocated resources for query execution.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 8

Use slides 6 to 8 to explain the steps involved in processing the SQL statements through JDBC API.

The process of executing a SQL query using JDBC is a six step process:

- Based on the database used, the application has to load appropriate JDBC driver. There is a driver for database such as one for Oracle, one for MySQL and so on. The JDBC driver provides connection to the database and helps in transferring queries and their results between Java application and relational databases.
There are different types of drivers:
 - JDBC-ODBC bridge driver
 - Native API party Java driver
 - NET-protocol All-Java driver
 - Vendor-specific driver
- Once required driver is loaded, a logical connection has to be established with the database. To establish the connection, either the `DriverManager` class or `DataSource` class can be used. The `DriverManager` class provides `getConnection()` method to establish connection with the database. A `DataSource` object can be used to connect to the database by setting its properties.
- JDBC provides different interfaces to SQL statements – `Statement` interface, `PreparedStatement` interface and `CallableStatement` interface.
The `Statement` interface creates SQL statements as constants. `PreparedStatement` interface implements precompiled SQL statements to which parameters can be passed. `CallableStatement` is used to implement procedures in SQL.
- The `Statement` interface provides methods for executing SQL queries. It has `execute()`, `executeUpdate()` and `executeQuery()` methods which can be used for executing queries.
The result of these queries is stored in a `ResultSet` object.
- JDBC processes the result of the query retrieved into `ResultSet` object using a cursor.
- Once the query is executed the logical connection established has to be closed.

Slide 9

Let us understand the limitations of JDBC API.

Limitations of JDBC API

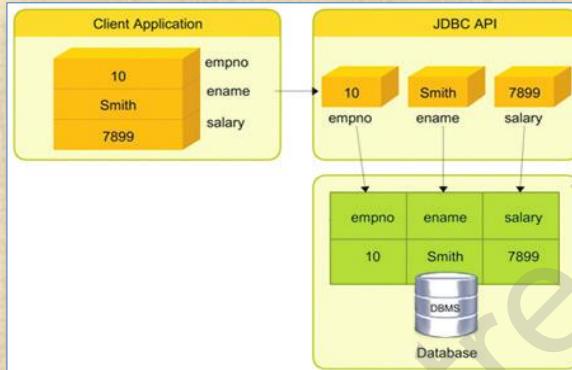






□ Following are the drawbacks of the JDBC API:

- Developers have to manually map the object representation to the relational database model in the JDBC code.
- JDBC application contains large amount of code which are database specific.
- Following figure demonstrates the storage of object using JDBC API:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 9

Use slide 9 to explain the limitations of JDBC API. Explain the drawbacks with the help of the following points mentioned in the slide 9. You can share your own experience or examples of using JDBC API and how it limits the use of JDBC API and what problems are faced by the developers.

- Developers have to manually map the object representation to the relational database model in the JDBC code.
- JDBC application contains large amount of code which are database specific.

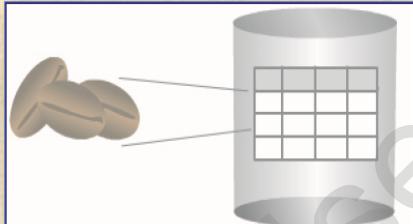
Slide 10

Let us understand the implementation of persistence in EJB 2.0.

Persistence with EJB 2.0 1-2



- EJB 2.0 introduced Entity Beans for data persistence.
- An Entity corresponds to a row in the relational database table.
- Data is accessed through Entity instances in the application.
- Whenever store or retrieve operation is performed, the data from the database is loaded in the entity instance created in the memory.
- Following figure shows Entity bean model:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 10

Use slide 10 to explain how persistence was implemented in EJB 2.0. It introduces Entity beans through which the data in the application database can be managed. Each instance of Entity bean represents a row of data in the database. With the help of slide 10, explain how the data is accessed by the Entity Beans from the database.

Slide 11:

Let us understand the features of Entity beans in EJB 2.0.

Persistence with EJB 2.0 2-2



Following are the features of the Entity Beans:

Persistence of data

Shared Access by application components

Primary key implementation

Relationships among various entities



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 11

Use slide 11 to explain the features of Entity beans. The main features of the Entity beans can be explained as follows:

Persistence

The Entity bean stores the state in data storage. Persistence refers to the fact that the Entity bean's state exists beyond the lifetime of the application or the Application Server. The persistent data remains intact even after the storage shuts down.

Shared Access

Multiple users can use the same Entity bean. To maintain data consistency, the Entity beans work within a transaction, which is also a part of EJB package.

Primary Key

An Entity bean supports a unique attribute. This unique attribute called a primary key in the database parlance enables the client to locate the exact Entity bean.

Relationships

Like the data of a relational database, the Entity beans are also related to each other. For instance, EmployeeBean and SalaryBean may be related to employee_id and one may contain the employee's personal details and the other may contain the employee's salary details.

Slide 12

Let us understand Object Relational Mapping.

Object Relational Paradigm 1-3

- Object Relational Mapping (ORM) is a technique of persisting objects onto the database.
- ORM automates the task of object persistence to the database.
- ORM can be done either manually or through ORM tools.
- ORM tools contain automatic mapper which generate DDL for the target platform.
- Oracle TopLink and Hibernate are examples of ORM tools.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 12

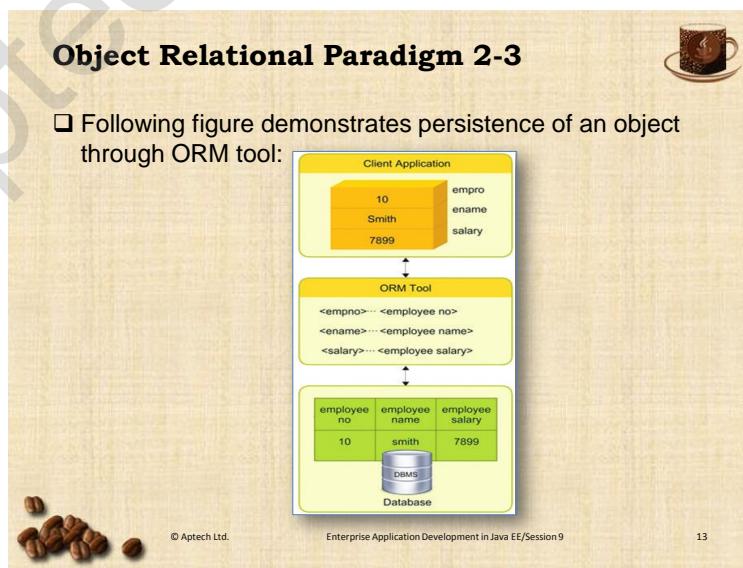
Use slide 12 to explain object relational mapping. Object relational mapping is the technique of mapping the objects in the object-oriented application to the relational database.

Object relational mapping can be done either programmatically or through ORM tools. Some of the tools available for object relational mapping are ActiveJPA, Athena Framework, and Fast Java ORM and so on.

The ORM tool contains an automated mapper that generates the data definition of the target platform using DDL. These data definitions are generated either from the Java class or from a descriptor file.

Slide 13

Let us understand object relational mapping graphically.



Use slide 13 to explain how the object data is converted into a row of relation through the ORM tool.

The object Employee in the bean has three variables empno, ename and salary. The ORM tool converts the object into a row in table employee where the columns Employee no, Employee name and Employee salary correspond to the attributes of the object.

Slide 14

Let us understand the working of the ORM tool.

Object Relational Paradigm 3-3

- The ORM tool contains an automated mapper that generates the data definition of the target platform using DDL.
- These data definitions are generated either from the Java class or from a descriptor file.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 14

Use slide 14 to explain the working of ORM tool. The tool has an automatic mapper which executes DDL statements such as data definition language statements to create the database schema onto which the entity data can be mapped.

The database schema is generated based on the class definition in the application. Java Class file or Descriptor file is used for the same.

Slides 15 and 16

Let us understand the various ORM tools.

ORM Tools 1-2



- Java Persistence API**
 - Manages persistence of Java objects by defining Entity classes
- Java Data Objects**
 - Defines object persistence through external XML meta files
- Hibernate**
 - Open source ORM framework which maps Java classes into database tables
- EclipseLink**
 - Open source ORM tool enabling interaction between application and other data services such as databases, Web services, EIS, and Object XML mapping

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 15

ORM Tools 2-2



- Fast Java Object Relational Mapping**
 - Lightweight ORM tool
 - Does not support mapping of m:n and 1:n relationships
- Java Object–Oriented Querying**
 - Object mapping software library implementing active record pattern
- Active JDBC**
 - An ORM tool based on active record pattern

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 16

Use slides 15 and 16 to explain the various ORM tools which can be used with Java applications.

Java Persistence API enables ORM, as it has certain predefined strategies that are used to map each entity class onto the database.

Java data objects define object persistence through external XML Meta files.

Hibernate is an open source ORM framework which maps Java classes into database tables. It also enables database querying and retrieval facilities. It is capable of generating SQL calls and manages the result set of handling the queries.

EclipseLink is an open source ORM tool which enables the interaction between the application and various other data services such as databases, Web services, Object XML mapping, and Enterprise Information Systems.

Fast Java Object Relational mapping is a lightweight ORM tool. It does not support mapping of m:n relationships and n:1 relationships.

Java Object Oriented Querying a database mapping software library which implements active record pattern.

Active JDBC is an ORM tool base in active record pattern.

Slide 17

Let us understand Java Persistence API.

Overview of Java Persistence API 1-2

- ❑ Was introduced in Java EE 5 as part of EJB 3.0 specification.
- ❑ Used to map relational data onto Java objects.
- ❑ Used for storage of Java objects onto relational databases.
- ❑ Defines a standard mechanism for ORM mapping.
- ❑ Manages data by converting the Plain Old Java Objects (POJOs) into entities.
- ❑ Current version of JPA in Java EE 7 is JPA 2.1.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 17

Use slide 17 to explain Java Persistence API. JPA was first introduced in Java EE 5 as part of EJB 3.0 specification.

JPA defines a standard mechanism of mapping Java objects onto relational databases. It is responsible for all the operations pertaining to database in the enterprise application.

The currently used version of JPA is JPA2.1 which was released as part of Java EE 7.

Slide 18

Let us understand the tasks carried out by Java Persistence API.

Overview of Java Persistence API 2-2

Java Persistence API (JPA)

- Defines a standard mechanism of object-relational mapping.
- Provides specification compliant products known as ORM tools for object-relational mapping.
- Allows inheritance among different entities of the application.
- Allows association of different entities.
- Manage applications' interactions with the database.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 18

Use slide 18 to explain the tasks performed by Java Persistence API.

Java Persistence API performs object relational mapping in enterprise applications. It also translates the inheritance in Java applications onto the database. It has certain in built strategies for translating the inheritance hierarchy in the application.

It establishes the required database connections, manages the communication and data retrieval from the database.

Slide 19

Let us understand the concept of entities.

Entities

- ❑ Persistent objects of enterprise applications are known as **entities**.
- ❑ Each entity is associated with data fields and methods.
- ❑ For example, `Bank_account` is an entity and it is associated with properties such as `account_number`, `account_holder_name`, and `account_balance`.
- ❑ An entity in the application domain represents a table in a relational database.
- ❑ Each row of data in the relation represents an entity instance in the application domain.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 19

Use slide 19 to explain the concept of entities in JPA. Entities correspond to the objects whose state has to be persisted in the application.

Each entity corresponds to an object in the application context; it has a well-defined state and behaviour. The state of the entity has to be persisted onto the database.

An Entity class in the application corresponds to a table in the database. An instance of the entity class corresponds to a row in the relational table.

Slide 20

Let us understand the requirements of the Entity class.

Requirements of Entity Class



- An Entity class should have a default constructor whose access specifier is protected or public.
- An Entity class should be annotated with javax.persistence.Entity.
- Neither the Entity class nor its methods should be final.
- An Entity class must implement Serializable interface, if it is passed by value to the business methods.
- Entity classes can be inherited.
- Persistence instance variables can be prefixed with access modifiers such as public, protected, or package private.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 20

Use slide 20 to explain the requirements of the entity class.

An Entity class has to satisfy the following requirements:

- An Entity class should have a default constructor whose access specifier is protected or public. Apart from a default constructor, the Entity class can have other constructors also.
- An Entity class should be annotated with javax.persistence.Entity.
- Neither an Entity class nor the methods and persistence variables of the Entity class should be declared final.
- Whenever an Entity instance is passed by value through business methods, then the entity class must implement Serializable interface.
- Like other Java classes, Entity classes can also be inherited. An Entity class can extend both entity and non-entity classes. A non-entity can also extend an Entity class.
- Persistence instance variables can be prefixed with access modifiers such as private, protected, or package private.

Slides 21 to 24

Let us understand Entity class in detail.

Entity Class 1-4

- Following code snippet shows an Entity class:

```
package Manage;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Message implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long message_id;
    private String text;
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 9

21

Entity Class 2-4

```
public Long getmessage_Id() {
    return message_id;
}

public void setmessage_Id(Long id) {
    this.message_id = id;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (message_id != null ?
message_id.hashCode() : 0);
    return hash;
}
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 9

22

Entity Class 3-4



```
@Override
    public boolean equals(Object object) {

        if (!(object instanceof Message)) {
            return false;
        }
        Message other = (Message) object;
        if ((this.message_id == null && other.message_id
        != null) || (this.message_id != null &&
        !this.message_id.equals(other.message_id))) {
            return false;
        }
        return true;
    }
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 9

23

Entity Class 4-4



```
@Override
    public String toString() {
        return "Manage.Message[ id=" + message_id +
    " ]";
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```



© Aptech Ltd.

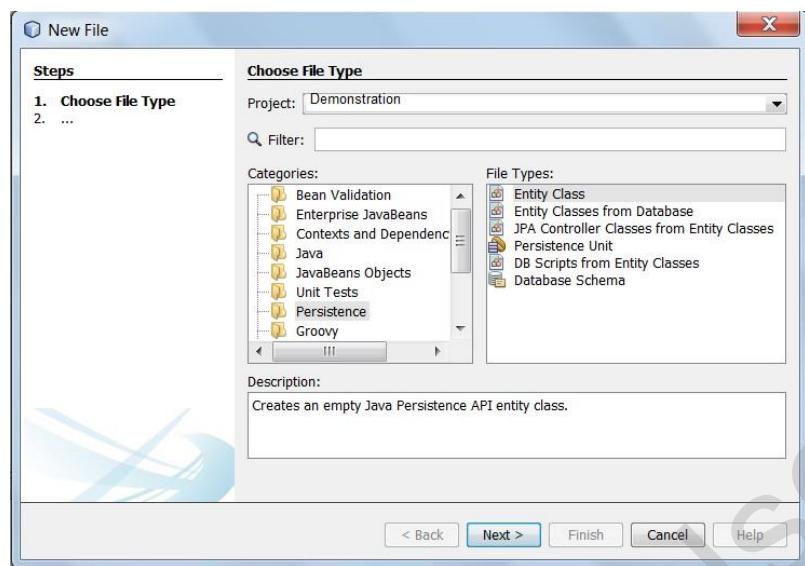
Enterprise Application Development in Java EE/Session 9

24

Use slides 21 to 24 to explain an Entity class. The slides demonstrate an Entity class Message, the message_Id variable of the class is mapped as the primary key onto the relational table. The Message entity is a String value, which is mapped onto the text field.

Every variable in the Entity class has to be encapsulated with getter and setter methods in the IDE. Demonstrate the process of creating the Entity class through the IDE.

Following image will show you how to create an Entity class.



This wizard will appear when the developer tries to add a new component to the EJB module in the application.

Tips:

The process of creating the Entity class is similar to that of creating a Session bean. Choose 'Entity class' while choosing the type of file to be created in the EJB module.

Slide 25

Let us understand primary keys in entities.

Primary Keys in Entities 1-3

- Every entity object is identified through a unique identifier known as primary key.
- Primary key can be a simple primary key or composite primary key.
- Simple primary key is annotated with `javax.persistence.Id`.
- Composite primary keys are defined in the primary key class and are annotated with `javax.persistence.EmbeddedId` and `javax.persistence.Id class`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 25

Use slide 25 to explain primary keys in entities and why it is necessary to use primary keys in every table created in the database and how it helps in creating, linking, and mapping the database objects using ORM.

Every piece of data in a database is identified through a primary key. All tables in the database must have a primary key value so that the values in the row can be uniquely identified.

The Entity class should therefore have a value which can be mapped onto the primary key column of the table. The primary key attribute has requirements such as it cannot have duplicate values. The Entity class should therefore annotate the primary key field to ensure that the variable does not assume duplicate values.

The annotation used for defining a primary key in the Entity class `javax.persistence.Id`.

A primary key can be a single value or a combination of more than one value. A primary key which is represented as a combination of multiple values is also known as composite primary key.

Applications can define the composite primary key as a primary key class. This class can be implemented as an embedded class in the Entity class. When the primary key is implemented as an Entity class then it has to be annotated with `javax.persistence.EmbeddedId`.

If it is represented as a separate primary key class, it has to be annotated with `javax.persistence.Id` class.

Slide 26

Let us understand the data types which can be used by the primary key in the Entity class.

Primary Keys in Entities 2-3

- Primary key field can be any one of the following data types:

Primitive data types	Java primitive wrapper classes	java.lang.String
java.util.Date	java.sql.Date	java.math.BigDecimal
java.math.BigInteger		

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 26

Use slide 26 to explain the data types which can be used by the primary key attributes in the Entity class. Explain each category of different types of data shown in slide 26.

Slide 27

Let us understand the requirements of the primary key class.

Primary Keys in Entities 3-3

- A primary key class has the following requirements:
 - Primary key class should have public access modifier.
 - Primary key class properties should be public or protected.
 - It must have a default constructor.
 - It must implement `hashCode()` and `equals(Object other)` methods.
 - It must be serializable.
 - Composite primary key class must be created as either embedded class or be mapped onto the properties of Entity class.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 27

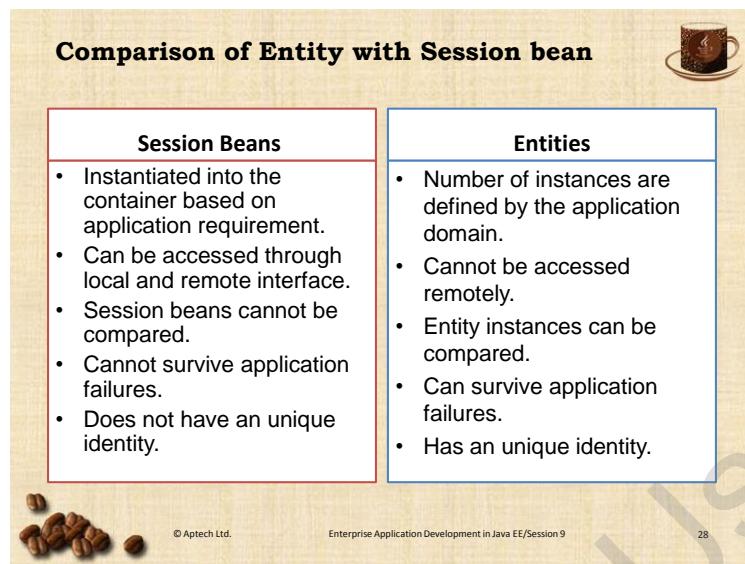
Use slide 27 to explain the requirements of primary key classes. Use the points given in slide 27 to explain all the requirements in order to create and use Primary Key Entity object.

- The primary key class should be publicly accessible. It should be prefixed with a public key word.
- The properties of the primary key class should be public or protected.
- The primary key class must have a default constructor.
- The primary key class must implement `hashCode()` and `equals(Object other)` methods.
- The class must be serializable.
- A composite primary key must be mapped onto the properties of the Entity class or created as an Entity class in the embeddable class.
- When the primary key class is mapped onto the properties of the Entity class then the corresponding names of the properties and type of the properties should be the same.

Slide 28

Let us compare Entity class with Session beans.

Comparison of Entity with Session bean	
Session Beans	Entities
<ul style="list-style-type: none"> Instantiated into the container based on application requirement. Can be accessed through local and remote interface. Session beans cannot be compared. Cannot survive application failures. Does not have an unique identity. 	<ul style="list-style-type: none"> Number of instances are defined by the application domain. Cannot be accessed remotely. Entity instances can be compared. Can survive application failures. Has an unique identity.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 28

Use slide 28 to explain the differences between Session beans and Entities in the application.

- Session beans are instantiated into the EJB container as per the application requirement. The numbers of instances to be created are decided by the container into which these beans are deployed.
- In case of entities, the number of instances of the entity class is defined by the application domain. Every entity instance is identified through domain specific data such as primary key. This identity can be used to access the entity or pass it as a reference to other application components.
- Session beans can be accessed locally or remotely through local interface and remote interface respectively. Entities cannot be remotely accessed.
- Different entities can be compared with each other and can be referenced through their respective unique identities (primary key). This is not the case for Session beans.
- The lifecycle of an entity is independent of the lifecycle of a Session bean. Entities can exist even after the application is shut down, thus having a longer life than the Session beans.
- Entities can survive critical application failures whereas Session beans cannot survive. The state of a Session bean is lost in case of system failure.

Slides 29 and 30

Let us understand the process of packaging and deploying Entities.

Packaging and Deploying Entity Classes 1-2



Entity classes are packaged as persistence units

A persistence unit is a set of logically connected entity classes

Persistence units are defined in a configuration file `persistence.xml`

Deployment descriptors are added to `META-INF` directory of the application



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 29

Packaging and Deploying Entity Classes 2-2



- ❑ Following code snippet shows the definition of a persistence unit in `persistence.xml` file:

```
<persistence>
<persistence-unit name="AccountOperation">
<description>This unit manages accounts of customers when the customer has multiple accounts this unit links these accounts </description>
<jta-data-source>jdbc/MyAccounts</jta-data-source>
<jar-file>MyAccount.jar</jar-file>
    <class>FixedDeposit</class>
    <class>SavingsAccount</class>
    <class>CurrentAccount</class>
</persistence-unit>
</persistence>
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 30

Use slides 29 and 30 to explain the process of packaging and deploying the Entity classes. The Entity classes of an application are packaged as a persistence unit.

A persistence unit comprises a set of connected Entity classes. A persistence unit can be mapped onto the database of the application. All the tables in the database are mapped onto the corresponding Entity classes in the persistence unit.

Each persistence unit has a deployment descriptor –`persistence.xml`. The deployment descriptor has the data connection URL and the jar file name (Java Archive file) into which the persistence unit is packaged.

The persistence unit shown in the deployment descriptor is `AccountOperation`, located at `jdbc/MyAccounts`. The persistence unit has three Entity classes – `FixedDeposit`, `SavingsAccount` and `CurrentAccount`.

Apart from the tags shown in the slide, the deployment descriptor may also have the following tags:

- `<provider>` which defines the persistence provider for the application. A persistence provider is responsible for managing permanent storage of data for the application.
- `<transaction-type>` defines the type of the transaction. This element can assume a value which is either `JTA` or `RESOURCE _ LOCAL`.
- `<mapping-file>` Object-relational mapping for the Entity classes can be specified through annotations or through the `orm.xml` file. The object-relational mapping can be specified through more than one mapping file. These mapping files are specified through `<mapping-file>` element.

Slide 31

Let us understand the persistence provider in detail.

Persistence Provider

Persistence provider refers to a third party technology which can be used to manage application data objects on the disk.
 MySQL and Oracle are some of the persistence providers.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 31

Use slide 31 to explain the concept of persistence provider. A persistence provider is a database provider that stores application data onto permanent storage. It provides a method of organising the data providers for implementing security mechanisms and allows defining backup mechanisms on the application.

Tips:

There are various persistence providers such as Oracle and MySQL that can be used with Java applications.

Slide 32

Let us understand how JPA can manage entities.

Managing Entities

- Entities in a persistence unit are managed through an EntityManager.
- EntityManager is an instance of javax.persistence.EntityManager.
- EntityManager**
 - Is responsible for managing activities associated with a set of entities such as:
 - persisting the entity
 - Removing an entity
 - Querying the entity
 - Each EntityManager instance is associated with a PersistenceContext.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 32

Use slide 32 to explain how JPA can manage entities.

Entities are managed through an instance of EntityManager. Each persistence unit is managed by an EntityManager. EntityManager is an instance of javax.persistence.EntityManager. An EntityManager is responsible for managing activities associated with a set of entities such as persisting the entity, removing an entity, and querying the entity. Every EntityManager instance is associated with a PersistenceContext instance.

Slide 33

Let us understand PersistenceContext.

PersistenceContext

- PersistenceContext keeps track of the state changes associated with an entity.
- PersistenceContext refers to a set of entities existing in an application environment.
- There are two variants of persistence context:
 - Transaction scoped persistence context – A set of entities associated with a transaction.
 - Extended persistence context – A set of entities independent of any transaction.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 33

Use slide 33 to explain PersistenceContext. PersistenceContext reflects the current state of the application database. It keeps track of any changes made to the entities in the PersistenceUnit.

The EntityManager when instantiated is associated with the PersistenceContext of the PersistenceUnit.

PersistenceContext provides an execution environment for running the queries on the application database.

Following are the two variants of the persistence context:

- Transaction scoped persistence context
- Extended persistence context

Transaction scoped persistence context refers to a set of entities associated with a transaction. The state of these entities may change during the course of transaction. If the transaction commits, the changes made to these entities are persisted. If the transaction rolls back, the changes made to the entities associated with the transaction are rolled back.

The lifetime of the persistence context is dependent on the lifetime of the transaction. Once the transaction commits and closes, any changes made to these entities are not persisted.

An extended persistence context is not dependent on the transaction. It is associated with the EntityManager instance and is managed manually by the application code.

Slide 34

Let us understand the EntityManager interface.

EntityManager Interface




© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 34

- ❑ EntityManager API is responsible for the following operations:
 - Creating and removing persistent entity instances.
 - Finding entities based on the primary key.
 - Allowing queries to run on entities.
- ❑ EntityManager instances can be created either through container or through application code.

Use slide 34 to explain EntityManager interface. An EntityManager is responsible for managing all the entities in the PersistenceUnit.

It provides a set of methods to create entities, remote entities, retrieve data from the persistence unit and so on. An application may have Container managed EntityManager or Application Managed EntityManager.

Slide 35

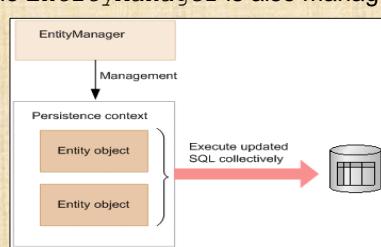
Let us understand container managed EntityManagers.

Container Managed EntityManagers




© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 35

- ❑ EntityManager is instantiated by the container.
- ❑ It defines PersistenceContext and shares it with all the application components.
- ❑ The lifecycle of the EntityManager is also managed by the container.



```

graph TD
    EM[EntityManager] -- Management --> PC[Persistence context]
    subgraph PC
        direction TB
        EO1[Entity object]
        EO2[Entity object]
    end
    PC -- "Execute updated SQL collectively" --> DB[Database]
  
```

Use slide 35 to explain container managed EntityManagers. The container creates an instance of EntityManager and associates it with PersistenceContext in the application. In this

case, all the components in the PersistenceContext are managed by respective EntityManagers.

Container also manages the lifecycle of the EntityManager.

Slide 36

Let us understand application managed EntityManagers.

Application Managed Entity Managers 1-3

- ❑ Application managed entity managers are used when the persistence context is not to be shared among all the application components.
- ❑ It creates an isolated PersistenceContext.
- ❑ Applications create EntityManager instance from an EntityManagerFactory class.
- ❑ createEntityManager() method of EntityManagerFactory class creates a thread-safe EntityManager instance.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 36

Use slide 36 to explain application managed EntityManagers. In this case the developer can decide when to instantiate and take services from the EntityManager.

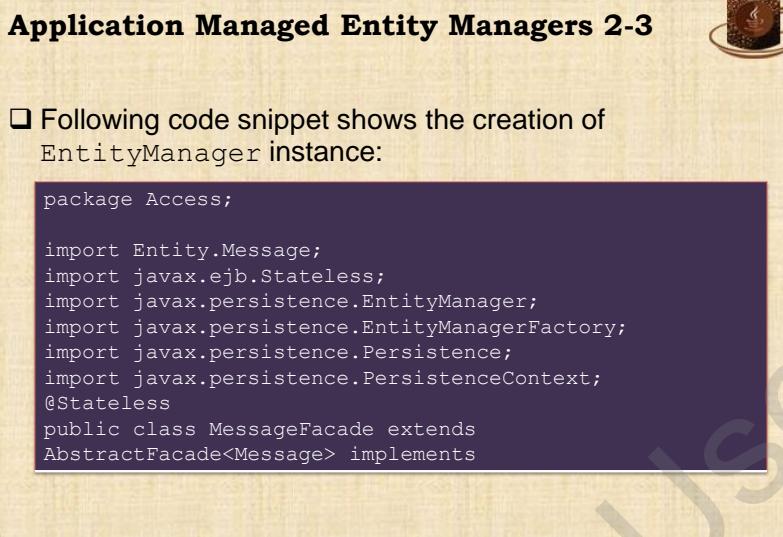
Container managed EntityManagers are used when the PersistenceContext can be shared among all the application components. When the PersistenceContext cannot be shared among all the application components, developers use application managed EntityManagers.

Application managed EntityManagers are associated with an isolated PersistenceContext.

An instance of EntityManager is created by invoking createEntityManager() method of EntityManagerFactory class.

Slides 37 and 38

Let us understand the instantiation of EntityManager in the application.

A large, semi-transparent watermark of the Java EE logo, which consists of the letters "Java" in a stylized font above the word "EE".

Application Managed Entity Managers 2-3

A small graphic of a coffee cup with steam rising from it, positioned in the top right corner.

□ Following code snippet shows the creation of EntityManager instance:

```
package Access;

import Entity.Message;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.PersistenceContext;
@Stateless
public class MessageFacade extends AbstractFacade<Message> implements
```

A small cluster of brown coffee beans located in the bottom left corner.

Application Managed Entity Managers 3-3

Use slides 37 and 38 to explain the implementation of EntityManager in an application.

Creating an EntityManager instance is not thread-safe. Therefore, applications create EntityManager instance from an EntityManagerFactory class. The createEntityManager() method of EntityManagerFactory class creates a thread-safe EntityManager instance.

An instance of the EntityManager can be obtained through the following statement:

@PersistenceUnit

```
EntityManagerFactory E;  
EntityManager EM = E.createEntityManager();
```

The code shown in the slide creates a PersistenceContext based on the persistence unit. The persistence unit used in this application is 'MessageStore-ejbPU'.

Create an EntityManager in the PersistenceContext with getEntityManager () method.

Slides 39 and 40

Let us understand the steps for creating an EntityManager and PersistenceContext.

Obtaining a Persistence Context 1-2

- The PersistenceContext is defined through the annotation @PersistenceContext.
- A PersistenceContext can be associated with a transaction.
- Following is the usage of PersistenceContext annotation:

```
@PersistenceContext(name = "PersistenceUnitName")
```




© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 39

Obtaining a Persistence Context 2-2

- PersistenceContext can also be associated with a Stateful Session bean.
- PersistenceContext is injected into Stateful Session bean through @PersistenceContext annotation.
- Following is the usage of @PersistenceContext annotation with a Stateful Session bean:

```
@PersistenceContext(unitName = "MessageStore-ejbPU")
```




© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 40

Use slides 39 and 40 to explain the process of obtaining PersistenceContext.

PersistenceContext refers to a set of managed entity instances which are managed by an EntityManager object. In a relational database, persistence context refers to the set of rows which are actively accessed. Every EntityManager is associated with a persistence context to manage a set of entities.

A PersistenceContext may be associated with a transaction or a Stateful Session bean.

It is injected into the Java class using the annotation @PersistenceContext.

When the `PersistenceContext` is transaction scoped, the existence of the persistence context is dependent on the life of the transaction.

An extended persistence context can be initiated in a Stateful Session bean. It is retained as long as the Stateful Session bean is in the container.

Slide 41

Let us understand the lifecycle of an Entity instance.

Managing the Lifecycle of an Entity Instance

- ❑ The lifecycle of an entity is managed through an `EntityManager`.
- ❑ An entity instance can exist in any one of the following states:

New
Managed
Detached

Removed

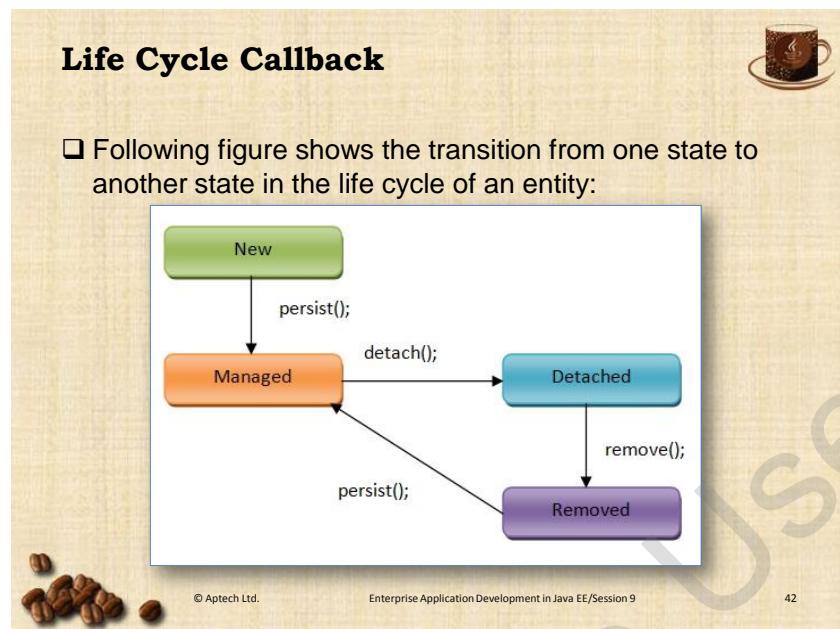
© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 41

Use slide 41 to explain different states in the lifecycle of an Entity instance. The lifecycle of Entity instance is managed through `EntityManager`.

- **New** – When the Entity instance is in this state, it does not have any persistence identity. It is also not associated with a persistence context. The `EntityManager` associates the entity with the `PersistenceContext`.
- **Managed** – When the Entity instance is in a managed state then the entity is associated with a `PersistenceContext` and is associated with an identity. A ‘New’ method can transform from ‘new’ state to ‘managed’ state when `persist()` method is invoked by the `EntityManager`.
- **Detached** – An Entity instance is said to be in a detached state if it is not associated with the `PersistenceContext`.
- **Removed** – An Entity instance is said to be in removed state when the entity is scheduled to be removed from the data store. The Entity instance transforms to ‘removed’ state when the `EntityManager` invokes a remove method on the managed entity.

Slide 42

Let us understand the lifecycle transitions of the Entity instance.



Use slide 42 to explain the transitions of the lifecycle of Entity instances.

The Entity instance moves from 'New' to 'Managed' state when a `persist()` method is invoked on it. If the entity is not associated with `PersistenceContext`, then the Entity is said to be in detached state. The instance would move from 'Managed' to 'Detached' state on executing `detach()` method.

Executing the `remove()` method removes the Entity instance and deallocates the memory.

Slide 43

Let us understand the function of persist () method.

persist() Method

persist() method stores the entity onto the database based on the type of persistence context in use.

Following are the uses of persist() method:

- In 'new' state – assigns an identity to the entity instance.
- In 'managed' state – the method invocation is ignored.
- In 'removed' state – the entity is transformed to 'managed' state.
- In 'detached' state – results in EntityExistsException.

If an entity references other entities then persist() method invocation is cascaded.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 43

Use slide 43 to explain usage of persist () method.

Following is the usage of persist() method:

- When persist() method is invoked on an entity in 'new' state, then the entity is assigned an identity. On invoking the persist() method, the entity is stored onto the database based on the type of persistence context in use.
- If the persist() method is invoked in 'managed' state, then the method is ignored. If the current entity references other entities in the application then the persist() method is cascaded to the referenced entities. The option of how the persist operation has to be cascaded to the referenced entities is defined in the deployment descriptor of the application.
- If the Entity instance is in 'removed' state and persist() method is invoked, then the entity transforms to 'managed' state.
- If the Entity instance is in detached state and persist() method is invoked, then it results in an EntityExistsException.

Slide 44

Let us understand the usage of `remove()` and `detach()` methods.

remove() and detach() Method

remove() method

- In 'new' state it is ignored
- In 'managed' state `remove()` method transforms the entity into 'removed' state
- In 'removed' state it is ignored
- In 'detached' state it throws an `IllegalArgumentException`

detach() method

- In all states the `detach()` method removes the entity from the `PersistenceContext`
- If the entity is already in 'detached' state then it results in `IllegalArgumentException`

© Aptech Ltd.
Enterprise Application Development in Java EE/Session 9
44

Use slide 44 to explain `remove()` and `detach()` methods.

Following is the usage of `remove()` method in different states of entities:

- When `remove()` method is invoked on an entity in 'new' state then it is ignored, however any other entities referenced by the current entity are removed. This removal is dependent on the option provided for the cascade operation.
- When `remove()` method is invoked on the entity instance when it is in 'managed' state then the entity transforms to removed state. This operation is cascaded onto all the referenced entities based on the options provided for the cascade operation in the deployment descriptor.
- When `remove()` method is invoked on an entity in 'detached' state, then it throws an `IllegalArgumentException`.
- If `remove()` method is invoked on an entity in 'removed' state, then the method invocation is ignored.
- `detach()` method is used to remove an entity from the persistence context. If the entity is already detached from the persistence context, then it will throw an `IllegalArgumentException`.

Slide 45

Let us understand how persistent objects are mapped onto the database.

Mapping the Persistent Objects

- ❑ Objects are persisted onto the relational database by the EntityManager
- ❑ The code snippet shows the BankAccount entity class:

```
...  
@Entity  
public class BankAccount implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    public Long getId() {  
        return id;  
    }  
    public void setId(Long id) {  
        this.id = id;  
    }  
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 45

Use slide 45 to explain how objects are persisted onto the relational database.

Enterprise applications store persistent data objects in relational databases. In the application, the data objects are created as Entity classes. The instances of the Entity classes are termed as entities. These entities are persisted onto the relations of a relational database.

The instances of the Entity class are persisted onto the database by the EntityManager. The EntityManager creates a new row in the relation and then stores the data of the entity onto the relation. The EntityManager should also manage the updates and modifications made to the entity during the application execution.

Slide 46

Let us understand the mapping of entities onto the database using deployment descriptor.

Mapping of Entities onto Database Through XML Files 1-3

- ❑ XML files can be used for object-relational mapping.
- ❑ `orm.xml` is the deployment descriptor used for Object Relational Mapping.
- ❑ It is located in the `META-INF` directory.
- ❑ Other `.xml` files can also be used for entity mapping, which are defined through `<mapping-file>` element in `persistence.xml`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 46

Use slide 46 to explain how entities are mapped onto databases through XML files.

To map entities onto the relational database, the XML file '`orm.xml`' is used.

The persistence provider looks in the `META-INF` directory of the enterprise application for the '`orm.xml`' file. Entity mapping can be provided through other `.xml` files also. The mapping file should be defined through the `<mapping-file>` element in the `persistence.xml` deployment descriptor.

Slides 47 and 48

Let us understand object relational mapping through deployment descriptor.

Mapping of Entities onto Database Through XML Files 2-3



Following code snippet shows the mapping of the BankAccount entity:

```
<?xml version="1.0" encoding="UTF-8" ?>
<entity-mappings>
    ...
    ...
    <entity class="BankAccount" name="BankAccount">
        <table name="Account"/>
        <attributes>
            <id name="Account_number">
                <generated-value strategy="TABLE"/>
            </id>
        ...
    </entity>
</entity-mappings>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 47

Mapping of Entities onto Database Through XML Files 3-3



```
.....
<basic name="account_holder_name">
    <column name="acc_hld_name" length="100"/>
</basic>
<basic name="acc_balance">
    </basic>
</attributes>
</entity>
</entity-mappings>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 48

Use slides 47 and 48 to demonstrate the object relational mapping through deployment descriptor.

Following are the XML elements used to define entity mappings onto the database:

- `<entity-mappings>` element is the top element within which all the entity mappings are defined.
- `<entity>` element is used to define the entity which has to be mapped onto a table in the relational database. The entity element has the name of the Entity class in the application which will be mapped onto a table in the database.

- <attributes> element has the properties of the entities which are supposed to be mapped onto the columns of the table of the database.
- <id> element can be used along with the <attributes> element to identify the column which has to be designated as the primary key in the table.
- <generated-value> strategy specifies the method of generating the values of the primary key column. The value can be defined explicitly while instantiating the entity or they can be auto incremented based on some seed value.
- <basic> element is used to define mapping of entity properties onto the columns of the table.

In-Class Question:

After you finish explaining object relational mapping through deployment descriptor, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Where is the deployment descriptors located in the application?

Answer:

We can find the deployment descriptor in META-INF directory.

Slide 49

Let us understand the annotations used for object relational mapping.

Mapping of Entities to Databases through Annotations

Following are the annotations used to define the mapping of the entity classes onto the database:



@Table



@Entity



@Column



Use slide 49 to explain the different annotations used for mapping entities on to the database. Classes in the application can use @Entity, @Table and @Column annotations which can be used for object relational mapping.

Slide 50

Let us understand the `@Table` annotation in the application.

@Table Annotation

- ❑ Used by the EntityManager to identify the table of the database.
- ❑ Following code snippet shows the usage of @Table annotation:

```
@Entity  
@Table(name = "BankAccount")
```
- ❑ The @Table annotation can have four attributes – name, catalog, relational schema, and uniqueconstraints.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 50

The `@Table` annotation is used by the EntityManager to identify the table of the database onto which the entity of the annotated class has to be mapped.

The `@Table` annotation can have four attributes – name, catalog, relational schema, and uniqueconstraints.

The name attribute represents the name of the table onto which the current Entity class has to be mapped.

The catalog attribute is used to identify the relational table catalog in which the table is located. Similarly, the schema attribute is used to identify the relational schema to which the current table belongs.

The uniqueconstraints attribute can be used to define the requirement of unique values in certain columns of the table.

Slide 51

Let us understand the usage of @Column annotation.

@Column Annotation

- ❑ Used to define how certain property of the entity class can be mapped
- ❑ The column annotation has the following attributes:
 - name
 - unique
 - nullable
 - Insertable and updatable
 - columnDefinition
 - length
 - scale and precision

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 S1

Use slide 51 to explain the usage of @Column annotation.

@Column annotation is used to define how certain property of the Entity class can be mapped onto the column of a table in the database.

The @Column annotation has 8 attributes – name, unique, nullable, insertable, updatable, columnDefinition, scale and precision.

The name attribute defines the name of the column onto which the current property of Java entity has to be mapped.

The unique attribute can assume a true/false value. If the attribute value is set to true, then a unique constraint is applied on the column of the table.

The nullable attribute can assume a true/false value. When the nullable attribute is set to true, then it implies that the column can have null values otherwise the column cannot have null values.

The attributes insertable and updatable implies that the columns do not have an auto increment or preset values. The values into the column can be explicitly inserted or updated. The primary key columns are not generally updatable as changing the value of primary key can lead to inconsistent state of the database.

The columnDefinition attribute can be used to define the types of column.

The length attribute is used along with the columnDefinition attribute. If the column is defined to be of VARCHAR type, then the length defines the length of the string that can be inserted into the column.

The scale and precision attributes can be used if the column is of INTEGER type.

Slides 52 and 53

Let us understand the mapping of primary key classes.

Mapping Primary Keys onto the Database 1-2



- Primary keys are used to uniquely identify the entities of the class.
- Primary keys can be implemented either as a single property or set of properties.
- When implemented as a single property, the property is annotated with `@Id` annotation.
- When the entity class requires generated values for the primary key, it is annotated with `@GeneratedValue`.
- Primary keys can be defined through primary key classes or embedded classes.
- Primary key classes can be annotated with `@IdClass`.
- Embedded primary key classes are annotated with `@EmbeddedId`.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 52

Mapping Primary Keys onto the Database 2-2



- Following code snippet demonstrates the usage of primary key classes:

```
public class BankAccount_PK{
    private long Account_number;
    private String acc_hld_name;
}
@Entity
@IdClass(BankAccount_PK.class)
public class BankAccount{
@Id
private long acc_num;
@Id
private String account_holder;
...
}
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 53

Use slides 52 and 53 to explain the mapping of primary keys onto database.

When the primary key is implemented as a single property of the Entity class then the property is annotated with `@Id` annotation.

When the Entity class requires persistence provider generated keys for the primary key value, then `@GeneratedValue` annotation should be used in addition to the `@Id` annotation.

Primary keys are implemented as classes. Primary key classes are annotated with `@IdClass`, when implemented as embedded classes the primary key classes are annotated with `@EmbeddedId`.

Slide 54

Let us summarize the session.

Summary

- ❑ Java applications use Java Database Connectivity (JDBC) and Java Persistence API (JPA) to connect to the database.
- ❑ JPA defines a standard mechanism for object-relational mapping.
- ❑ JPA interprets the database objects in the application as entity classes.
- ❑ JPA defines an EntityManager to manage the entities in the application.
- ❑ The primary key for the entity objects can be implemented as an attribute in the entity class, as a primary key class or as an embedded class.
- ❑ orm.xml and persistence.xml files have the entity mapping information in the application.
- ❑ Persistent objects can be mapped using the declarative XML files.
- ❑ The persistent objects can be mapped onto the persistence unit by adding appropriate annotations to the Java classes.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 9 54

Using slide 54, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

9.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Advanced Persistence Concepts that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 10 – Advanced Persistence Concepts

10.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

10.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain how relationships are managed in OOP
- Explain how relationships are managed in relational databases
- Explain cardinality and directionality in object relationships
- Explain how relationships are managed in JPA
- Describe annotations provided by JPA to create object relationships
- Describe the mapping of different aspects of database to the enterprise application
- Describe the different JPA strategies to map inheritance in relational databases
- Explain implementation of inheritance among the entities

10.1.2 Teaching Skills

To teach this session successfully, you should be aware of about mapping of relationships in an object-oriented applications as well as in relational databases. You should prepare yourself on how manage relationships between the entities using JPA. Then, you should also prepare yourself on how to manage inheritance on the entities in the Java enterprise application.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❑ Explain how relationships are managed in OOP
- ❑ Explain how relationships are managed in relational databases
- ❑ Explain cardinality and directionality in object relationships
- ❑ Explain how relationships are managed in JPA
- ❑ Describe annotations provided by JPA to create object relationships
- ❑ Describe the mapping of different aspects of database to the enterprise application
- ❑ Describe the different JPA strategies to map inheritance in relational databases
- ❑ Explain implementation of inheritance among the entities

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 2

Tell them that they will be introduced to mapping the relationships among different entities in the application database onto objects and interactions among these objects. The concepts of cardinality and directionality of the relationships among objects in the application will be explained.

The mechanisms provided by JPA to model these relationships among the applications will be introduced but will be explained in the session. They will also learn different strategies to map complex relationships among entities in the Java enterprise applications using JPA technology.

10.2 In-Class Explanations

Slide 3

Let us understand entity relationships.

Introduction

- ❑ Entity beans in an enterprise application usually relate to one another.
- ❑ For instance, the Student entity bean is related to the Teacher entity bean in an enterprise application because the students are taught by the teacher.
- ❑ Implementing the relationship between the student and teacher is handled differently by application developers and database designers.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 3

Use slide 3 to explain entity relationships.

Tell them that the entities created in an enterprise application are normally related to each other. For instance in the context of University, **Teacher**, **Student**, and **Subject** are entities. The **Student** entity is related to the **Teacher** entity in an enterprise application because the students are taught by the teacher. Similarly, the **Student** is related to the **Subject** entity by the type of course they select in the university.

Now, the way the relationships are managed in the programming languages and in the relational databases is altogether different techniques.

Discuss with them how they will manage the relationships between **Teacher**, **Student**, and **Subject** entities in a database. Similarly, ask them, if the same entities were classes in an object-oriented application, then how they would create relationships between these classes.

After you receive the appropriate solutions for the discussion on managing relationships in databases and object-oriented programming languages, bring the approach they followed to achieve so.

Tell them, implementing the relationship between the student and teacher is handled differently by application developers and database designers. Today, most of the languages are based on object-oriented approach, which deals with objects and classes. The relational databases deal with tables and key constraints to establish the relation between the data stored in the tables. Hence, the concepts and techniques used for managing relationships will also differ.

Slides 4 to 7

Let us understand the relationships among objects in the application.

Relationships in Object-Oriented Programming 1-4



For Aptech Course Only

- ❑ Objects interact with other objects to represent some concrete function.
- ❑ There are three types of association among the objects:
 - Association
 - Aggregation
 - Inheritance
- ❑ Relationships among objects describe how objects collaborate, to contribute to the behavior of the system.

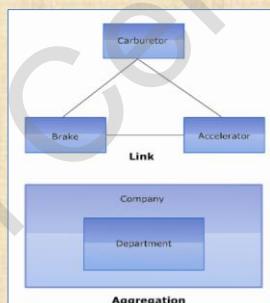
© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 4

Relationships in Object-Oriented Programming 2-4



For Aptech Course Only

- ❑ Nature of a relationship can be broadly classified as, link and aggregation.
- ❑ Following figure depicts relationships among objects:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 5

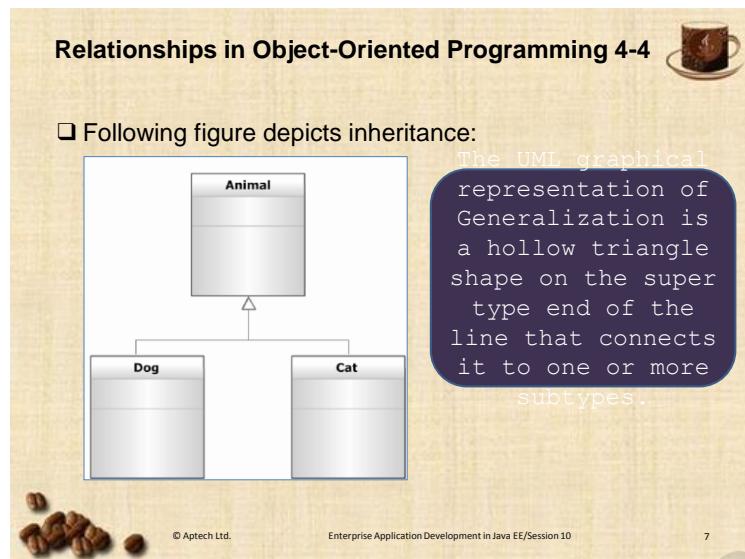
Relationships in Object-Oriented Programming 3-4



For Aptech Course Only

- ❑ In generalization, an object or subtype is dependent on another object or super type.
- ❑ Generalization relationship is used to reuse attributes, operations, and relationships present in the super type with one or more subtypes.
- ❑ The generalization relationship is also known as inheritance or 'is-a' relationship.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 6



Use slides 4 to 7 to explain different relationships among the objects created in the application context.

There are three types of relationships among the objects in an application:

- **Association** – It is a relationship between two objects, an object of one type makes use of properties of another object's operations.
- **Inheritance** - It is a relationship among application objects where there is a hierarchy among different classes of objects. In case of inheritance a class acquires properties of another class. One class of objects are said to be derived from another class of objects. Multiple classes can be derived from a single class.
- **Aggregation** - It is a relationship among objects where a class of objects inherits properties from a set of objects.

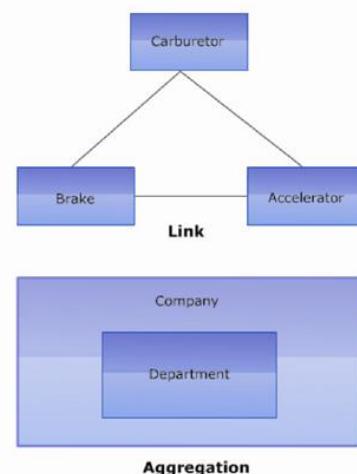
Various objects in the application collaborate with each other to define the behaviour of the system.

Use slide 5 to explain different types of relationships in an object-oriented application.

Nature of a relationship can be broadly classified as, link and aggregation. A link refers to a connection between two classes of objects; there may not be any common properties between the two classes of objects.

Aggregation refers to one object contained within another object; there are common features among the two types of objects.

Following figure shows the relationships between the objects:



Use slide 6 to explain another category of relationships among the objects of the application. It is generalization and specialization. These relation types are applicable to those classes which are part of an inheritance hierarchy. The relation between a superclass and its subclass is referred to specialization, that is, a subclass is a specialization of the superclass.

Similarly, the relation between the subclass and superclass is generalization. A superclass is a generalized version of a subclass.

Use slide 7 to explain the concept of generalization and specialization. As shown in the figure on slide 7, the **Animal** class is a generalization of the classes **Cat** and **Dog** classes. Similarly the class **Cat** and class **Dog** are specialization of the super class **Animal**.

Tell them that the UML graphical representation of Generalization is a hollow triangle shape on the super type end of the line that connects it to one or more subtypes. The Generalization relationship is also known as **inheritance** or 'is-a' relationship.

Slides 8 and 9

Let us understand the concept of object modelling in the application.

Object Modeling 1-2



Object modeling is widely done through class diagrams. UML diagrams are used to depict objects and relationships in an object model. Uses various notations to represent the following entities in diagrammatic form:

- Classes
- Attributes
- List of operations
- Access modifiers

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 8

Object Modeling 2-2



Following figure depicts a class diagram:

Class
Attribute
Operation()
Rectangle
- length:int
- width:int
+ getArea()
+ getLength()
+ getWidth()

- The topmost section contains the name of the class.
- The middle section contains a list of attributes.
- The bottom section contains a list of operations.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 9

Use slides 8 and 9 to introduce the concept of object modelling in the application.

Object modelling is part of application design phase. The application domain has been modelled into classes, objects, and interaction between the classes and objects.

The application designers have to define these aspects of the application, based on which the developers write code and create the application.

Application designers use UML diagrams for object modelling of the application. UML stands for Unified Modelling Language. It is a graphical language used for object modelling. It has graphical notations for representing classes, objects and their mutual relationships.

Use slide 9 to explain the notations used in class diagram.

A class diagram follows a simple notation. A class is represented as a rectangle divided into three sections. The topmost section contains the name of the class. The middle section contains a list of attributes, and the bottom section contains a list of operations or methods in the class. Access modifiers can also be represented in a class diagram. The '+' symbol is used for public, '-' is used for private, and '#' is used for protected.

Slides 10 and 11

Let us understand the concept of multiplicity in applications.

Multiplicity in Relationships 1-2



Relationships among objects are shown using connectors. Multiplicity in relationships is depicted using the following notations:

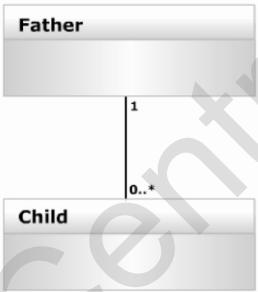
- 0..1 No instances, or one instance
- 1 Exactly one instance
- 0..* or * Zero or more instances
- 1..* At least one instance

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 10

Multiplicity in Relationships 2-2



Following figure depicts multiplicity in a relationship:



```

classDiagram
    class Father {
        <<1>>
    }
    class Child {
        <<0..*>>
    }
    Father "1" -- "0..*" Child
  
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 11

Use slides 10 and 11 to explain the concept of multiplicity in relationships.

When two classes are related to each other, the number of objects of one class to be associated with the number of objects of another class may vary.

For instance, when we consider a class **Teacher** and class **Student** and the relationship among these two classes is teaches that according to the relationship, an object of type **Teacher** might be associated with multiple instances of the class **Student**. Such a relationship is termed as **one-is-to-many** relationship.

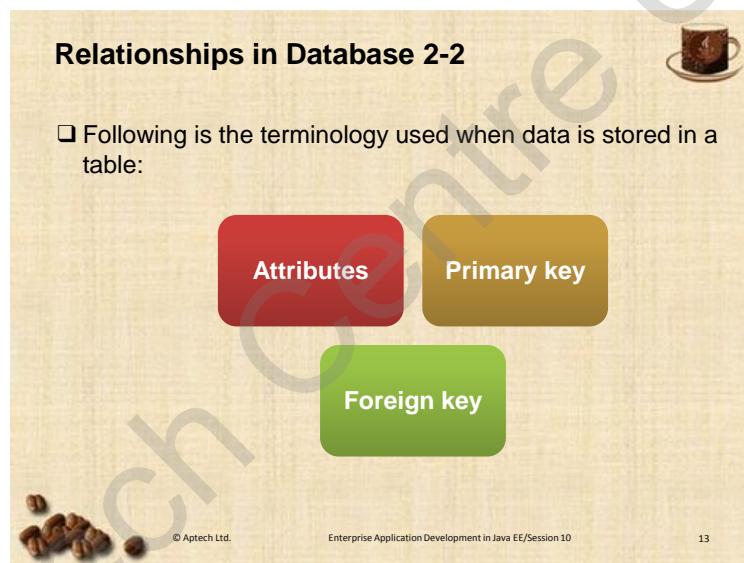
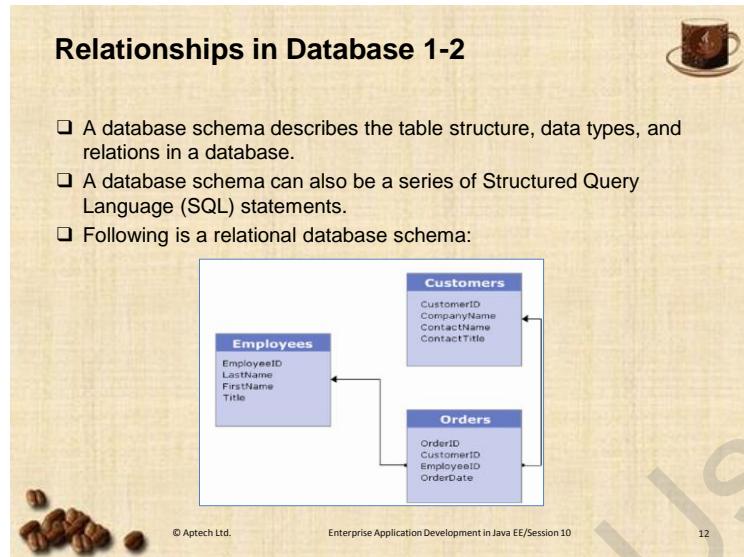
Multiplicity in class diagrams is depicted by labelling the connectors of the class. You can also depict multiplicity in relationships using the following notations at each ends of connectors:

- 0..1 No instances, or one instance
- 1 Exactly one instance
- 0..* or * Zero or more instances
- 1..* At least one instance

Use slide 11 to provide an example of a **one-to-many** relationship. The class Father and the class Child are related through a one-to-many relationship where one instance of Father can be associated with many instances of the Child class.

Slides 12 and 13

Let us understand how to manage the relationships in a database.



Use slides 12 and 13 to explain how to manage relationships in a database.

In a relational database all the data is structured in the form of tables. Each table has data organised in the form of rows and columns.

Each table represents an aspect of the application data. Relational database has a corresponding Structured Query Language (SQL) which can execute commands on the relational database and manipulate the database.

A database schema represents the structure of the database, the pattern according to which data in the application is organised.

Slide 12 shows an example database schema comprising three tables – **Employees**, **Customers**, and **Orders**. There is a relationship between **Orders** and **Employees** table and there is a relationship between **Orders** and **Customers** table.

Use slide 13 to introduce important terms used in relational databases. Here, we introduce three terms – **Attributes**, **Primary key**, and **Foreign key**.

- Attributes are properties of an entity. Each entity is associated with more than one attributes.
- Primary key is one of the attributes associated with the entity, a primary key always has unique values. This attribute is used to uniquely identify every row in the table.
- Foreign key is an attribute of the entity which is used to represent a relationship between two entities.

In-Class Question:

After you finish explaining how to manage relationships in database, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



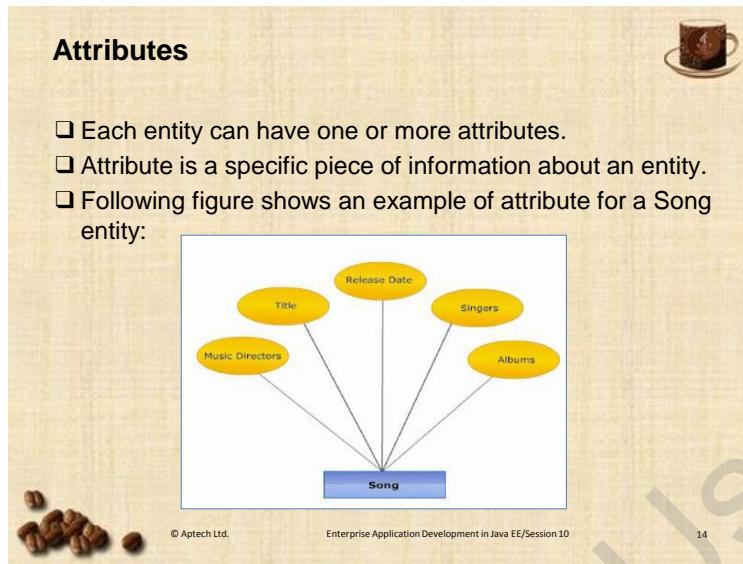
Which constraint in the database uniquely identify every row in the table?

Answer:

Primary key

Slide 14

Let us understand attributes.



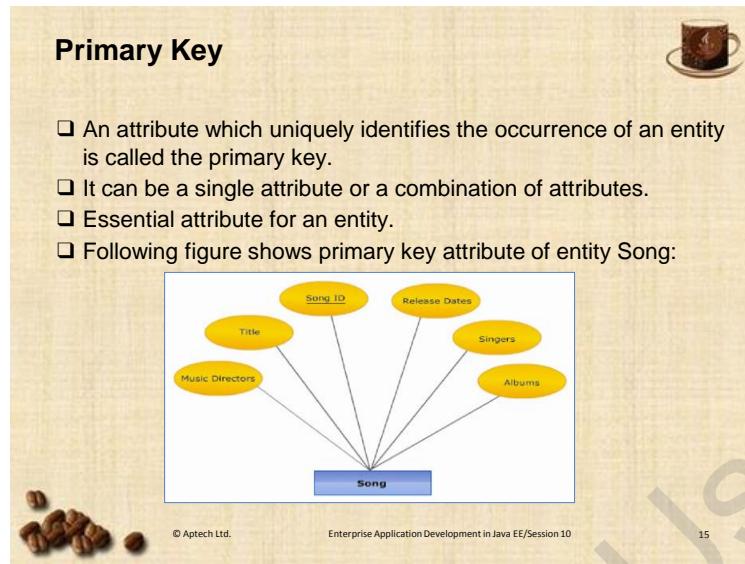
Use slide 14 to explain attributes in relational database.

Attributes are the properties associated with an entity in the database. An entity is represented as a rectangle and its corresponding attributes are represented with ellipses. Each attribute corresponds to a column in the table.

The entity Song has attributes **Music Directors**, **Title**, **Release Date**, **Singers**, and **Albums**.

Slide 15

Let us understand primary keys.



Use slide 15 to explain a primary key in a table.

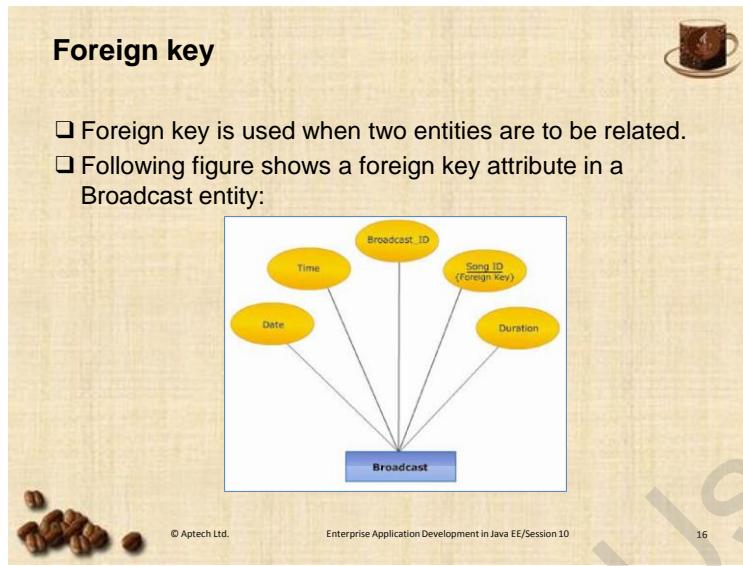
When large amount of data is stored in the database, we require mechanism to retrieve every value from the database. Primary key provides this mechanism to identify the record on the unique data.

Primary key always have unique values in the table, this attribute cannot have repetitive values in the column.

A primary key is a single attribute or a combination of attributes that uniquely defines an entity instance. A primary key is always a mandatory attribute of an entity. An entity cannot exist without the primary key. If such an attribute does not exist naturally, a new attribute is defined for that purpose, for example, an ID number or code. For a Song entity, the song title cannot be the primary key as there may be more than one song with the same title, so it requires an ID attribute.

Slide 16

Let us understand foreign keys.



Use slide 16 to explain foreign keys.

A foreign key is an attribute which enables you to define a relationship between two tables.

A foreign key in a table is a primary key in the table to which it is related.

If the primary key of one entity is available as an attribute of another entity, then it qualifies as a foreign key. When considering the two entities; Song and Broadcast, the Song ID primary key also finds a place as an attribute in the Broadcast entity, where it is the foreign key. Like primary key, the foreign key may consist of one or more attributes.

Slide 17

Let us introduce different types of relationships in a database.

Relationship

- ❑ Foreign keys help in forming relationships between entities.
- ❑ There are four types of relationships that can be created between entities in the database:
 - One-to-one relationship
 - One-to-many relationship
 - Many-to-one relationship
 - Many-to-many relationship

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 17

Use slide 17 to explain different types of relationships through which different entities in a database can be related.

List the four types of relationships created between the entities in the database.

Tell them that in One-to-one relationship between two types of entities, one entity of a class is related with only one entity of another class. In One-to-many relationship, one entity of a type can be associated with many entities of the other type.

Similarly in a many-to-one relationship, many entities of a certain type are associated with one entity of the other type. In case of many-to-many relationships, each entity of a type is associated with many entities of the other type and vice versa.

Following figure shows the one-to-one relationship between two entities:



Slides 18 and 19

Let us understand the concept of data modelling in databases.

Data Modeling 1-2



 © Aptech Ltd. Enterprise Application Development in Java EE/Session 10 18

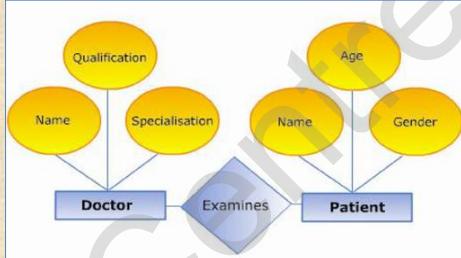
- ❑ Data modeling is usually achieved with the help of an Entity-Relationship Diagram.
- ❑ Data modeling concepts such as entities, attributes, and relations are represented in the ER diagram using symbols.
- ❑ To create an ER diagram, you should identify the entities, attributes, and relationships of an activity for which data has to be maintained.

Data Modeling 2-2



 © Aptech Ltd. Enterprise Application Development in Java EE/Session 10 19

- ❑ Following figure shows an ER diagram:



Use slides 18 and 19 to introduce the concept of data modelling in databases.

Like object modelling for object-oriented applications, we require data modelling for databases. Data modelling organises the data to be stored in the database into tables, their respective attributes and relationships among the attributes.

Database schema and E-R diagrams can be used for data modelling. E-R diagrams such as class diagrams are graphical representations of data to be stored in the database. We represent entities, their attributes and their relationships through E-R diagram.

Use slide 19 to demonstrate the E-R diagram. The E-R diagram has two entities – Doctor and Patient.

The entity **Doctor** has three attributes – **Name**, **Qualification**, and **Specialization**.

The entity **Patient** has three attributes – **Name**, **Age**, and **Gender**.

The entities **Doctor** and **Patient** are related through the relationship **Examines**.

Slide 20

Let us understand how to manage entities relationships in JPA.

Managing Entities Relationship in JPA

The JPA specification provides support to the following aspects of entity relationships:

- Entity inheritance
- Polymorphism
- Managing relationships and associations
- Polymorphic queries

Entities in the enterprise applications can be associated based on two things:

- Cardinality
- Directionality

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 20

Use slide 20 to explain how to manage relationships in JPA.

Entity beans represent objects in Object-Oriented Analysis and Design (OOAD). Data modeling and object modeling are the two most important aspects of OOAD, as they help in optimally designing your application. Object modeling is arranging a standardized set of symbols to represent the design of the system. Data modeling enables to structure and organize data so that it is implemented in a database.

In order to support entity relationships, the object-to-relational mapping engine must provide support for object-oriented features such as inheritance, polymorphism, and so on.

The Java Persistence specification supports the entity inheritance and polymorphism, managing relationship associations, polymorphic queries, and so on. The association among the entities is also termed as relationships among entities.

Entities in the enterprise applications can be associated based on the two things:

- Cardinality
- Directionality

Tips:

Earlier, EJB 2.0 provided entity beans to implement database objects in the Java enterprise applications. However, the implementation of the entity beans were complex. Each entity bean was accompanied by a Home interface and a component or remote interface along with the bean implementation class. The bean class was implementing the `EntityBean` interface. The EJB 2.0 specification enables the relationships between entity beans through container-managed or bean-managed. However, the process was complex to achieve relationships between two entity beans.

Slides 21 to 23

Let us understand the concept of cardinality.

Cardinality 1-3




© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 21

- ❑ The number of instances of an entity bean that relates to the number of instances of another bean is cardinality.
- ❑ JPA supports three types of cardinality relationships:
 - **One-to-one**
 - In one-to-one relationship, one bean instance relates to only one bean instance.
 - Relationship between a student bean and score card bean is an example of one-to-one relationship, as one student can be related to only one score card.

Cardinality 2-3




© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 22

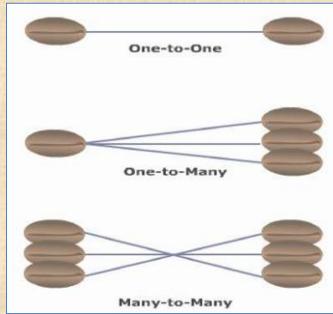
- **One-to-Many/Many-to-One**
 - In one-to-many relationship, one bean instance relates to multiple bean instances.
 - For instance, one customer bean instance can relate to multiple invoice beans but one invoice cannot be related to multiple customers.
- **Many-to-Many**
 - In many-to-many relationship, many bean instances relate to many instances of another bean.
 - The fact that many actors work in many movies is an instance of many-to-many relationship.

Cardinality 3-3




© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 23

- ❑ Following figure demonstrates different types of cardinality:



One-to-One

One-to-Many

Many-to-Many

Use slides 21 to 23 to explain the concept of cardinality.

Use slide 21 to explain the cardinality that represents the number of entities of one type which are related with the number entities of the other type.

JPA supports three cardinality relationships:

- One-to-one
- One-to-many/Many-to-one
- Many-to-many

In one-to-one relationship, one bean instance relates to only one bean instance. Relationship between a student bean and score card bean is an example of one-to-one relationship, as one student can be related to only one score card.

Use slide 22 to explain the other types of relationships supported by JPA.

In one-to-many relationship, one bean instance relates to multiple bean instances. For instance, one customer bean instance can relate to multiple products ordered on shopping cart. Similarly, multiple products ordered by the customer can also be shown. This type of relational is bi-directional.

In many-to-many relationship, many bean instances relate to many instances of another bean. The fact that many actors work in many movies is an instance of many-to-many relationship.

Use slide 23 to explain the figure that shows the graphical representation of the different types of relationships supported by JPA.

Slide 24

Let us understand directionality.

The slide has a light beige background with a subtle wood-grain texture. At the top left, the word "Directionality" is written in a bold, black, sans-serif font. In the top right corner, there is a small icon of a coffee cup filled with dark coffee, sitting on a saucer. At the bottom left, there is a small cluster of brown coffee beans. The bottom right corner contains some small, faint text: "© Aptech Ltd.", "Enterprise Application Development in Java EE/Session 10", and "24".

Directionality defines the navigation pattern between two beans.
 The navigation pattern can be unidirectional or bidirectional.

Use slide 24 to explain the directionality in relationships.

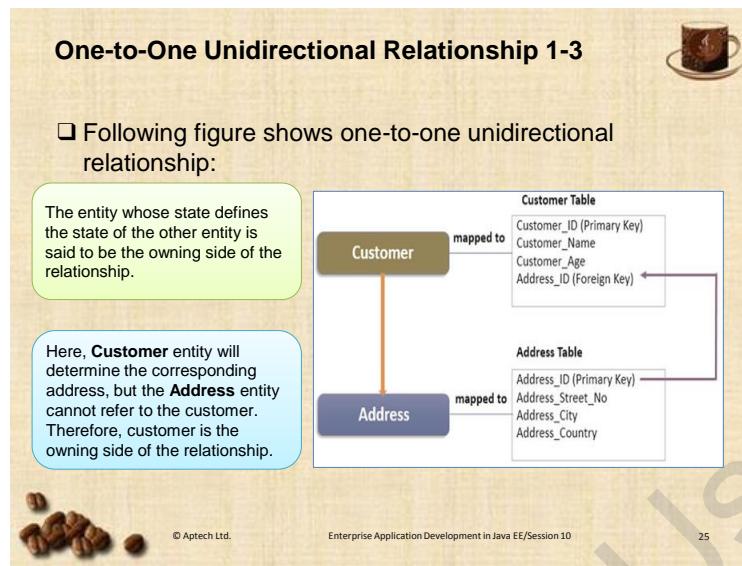
A relationship can be termed bi-directional based on the navigation pattern between the entities.

For instance the relationship between a **Teacher** entity and **Student** entity is a unidirectional relationship from **Teacher** and **Student**, but this relationship cannot hold in the opposite direction.

Thus, every relationship can be associated with a direction which defines the navigation pattern between two beans. Directionality can be either unidirectional or bidirectional.

Slide 25

Let us understand one-to-one relationship with the help of an example.



Use slide 25 to demonstrate an example of one-to-one relationship.

In a one-to-one relationship, the state of one entity determines the state of another entity. The entity whose state defines the state of the other entity is said to be the owning side of the relationship.

Consider two entities **Customer** and **Address**, the **Customer** entity will determine the corresponding address, but the **Address** entity cannot refer to the customer. Therefore, customer is the owning side of the relationship.

Slide 26

Let us understand annotations provided by JPA to implement one-to-one relationship.

One-to-One Unidirectional Relationship 2-3

- ❑ One-to-one relationships are mapped using primary key and foreign key associations.
- ❑ They are annotated through `javax.persistence.OneToOne`.
- ❑ `@OneToOne` annotation has the following attributes:
 - `targetEntity`
 - `cascade`
 - `fetch`
 - `mappedBy`
 - `orphanRemoval`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 26

Use slide 26 to explain how JPA implements one-to-one relationship.

In JPA, entity classes associated with one-to-one relationships are annotated through `javax.persistence.OneToOne`. The annotation is prefixed to the owning entity class of the application. The developer can define a one-to-one mapping either at the field level or in the getter/setter methods defined for the property.

The `@OneToOne` annotation has attributes such as `targetEntity`, `cascade`, `fetch`, `mappedBy`, and `orphanRemoval`.

- `targetEntity` represents the entity class to which the Entity is associated.
- `cascade` attribute defines how certain operations can be percolated through the relationship to the associated entities. By default, no operations are cascaded.
- `fetch` attribute defines the pattern of loading the entities involved in the association. This fetching mechanism of the entity can be either lazy or eager. According to the lazy fetching mechanism, when an entity is fetched all the entities associated with the entity are not fetched. In case of eager fetching mechanism when an entity is fetched by the application, all the associated entities are also fetched. The **EAGER** strategy is a requirement on the persistence provider runtime that the associated entity must be eagerly fetched. The **LAZY** strategy is a hint to the persistence provider runtime.
- `mappedBy` attribute is used to define a bidirectional relationship. It identifies the entity which will map the current entity through `mappedBy` attribute. This element is only specified on the inverse (non-owning) side of the association.
- `orphanRemoval` attribute is used to determine whether the removal of entity will result in the removal of the entities referred by it.

Slide 27

Let us understand the implementation of one-to-one relationship through JPA

One-to-One Unidirectional Relationship 3-3

Following code snippet demonstrates the one-to-one mapping for Customer and Address entities:

```

@Entity
@Table(name="Customer")
public class Customer {
    @Id
    @Column(name="Customer_ID")
    protected String Customer_ID;

    // Mapping Foreign Key
    @OneToOne
    @JoinColumn(name="Cus_address_id",
                referencedColumnName="Address_ID", updatable=false)
    protected Address address; // reference of Address object
}

@Entity
@Table(name="Address")
public class Address {
    @Id
    @Column(name="Address_ID")
    protected String Address_ID;
    ...
}

```

The `@JoinColumn` annotation contains the attribute `name` which refers to the name of the foreign key in the `Customer` table. The attribute `updatable` is set to `false`, which means that the persistence provider would not update the foreign key, even if the address reference were changed.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 27

Use slide 27 to explain the implementation of one-to-one relationship through annotations provided by JPA.

The `@JoinColumn` annotation contains the attribute `name` which refers to the name of the foreign key in the `Customer` table. The attribute `updatable` is set to `false`, which means that the persistence provider would not update the foreign key, even if the address reference were changed.

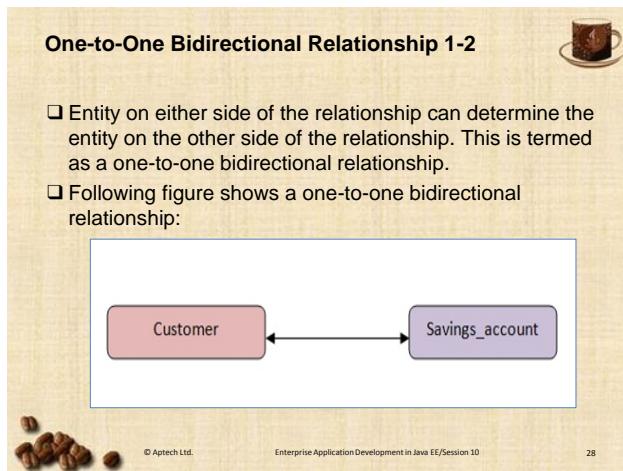
Additional References:

For more code Snippet on one-to-one relationship, you can refer the following link:

<http://www.mastertheboss.com/jboss-frameworks/hibernate-jpa/or-mapping/one-to-one-hibernatejpa-example>

Slide 28

Let us understand bidirectional one-to-one relationship.



Use slide 28 to explain bidirectional one-to-one relationship.

In a one-to-one bidirectional relationship, entities on either side of the relationship can be used to refer the other entity.

Consider a scenario of a bank application where a customer is assigned to a customer id. The domain also defines a constraint that every customer can hold only one savings bank account with the bank.

In such a scenario, with the help of customer id, the savings bank account number can be determined. Similarly, through the savings bank account number, the customer id of the account holder can also be determined.

Such a relationship is termed as bidirectional one-to-one relationship.

Tips:

If the relationship is bidirectional, the non-owning side must use the `mappedBy` element of the `OneToOne` annotation to specify the relationship field or property of the owning side.

Following code Snippet shows one-to-one association that maps a foreign key column between `Customer` and `CustomerRecord` class:

```
// On Customer class:
@OneToOne(optional=false)
@JoinColumn(
    name="CUSTREC_ID", unique=true, nullable=false,
    updatable=false)
public CustomerRecord getCustomerRecord() { return
customerRecord; }

// On CustomerRecord class:
@OneToOne(optional=false, mappedBy="customerRecord")
public Customer getCustomer() { return customer; }
```

Slide 29

Let us demonstrate the implementation of bidirectional one-to-one relationship.

One-to-One Bidirectional Relationship 2-2



Following code snippet shows the bidirectional mapping of Customer and Address entity:

```

@Entity
@Table(name="Address")
public class Address {
    @OneToOne(mappedBy="address")
    protected Customer customer;

    @Id
    @Column(name="Address_ID")
    protected String Address_ID;
    ...
}

```

The mappedBy element identifies the corresponding association field on the owing side of the relationship.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 29

Use slide 29 to demonstrate the implementation of one-to-one bidirectional relationship through code.

Explain the code Snippet showing the mapping of bidirectional mapping of Address entity class with Customer entity class. The `mappedBy` element identifies the corresponding association field on the owing side of the relationship. The value specified in this attribute is the relationship field defined in the Customer table.

In-Class Question:

After you finish explaining one-to-one relationships, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Consider a scenario where one-to-one association is created between a Customer and Address entities. Now, if you want that when a Customer is created, then the corresponding Address object is also created for persistence. Which attribute of `@OneToOne` annotation will help you to achieve this?

Answer:

`CascadeType`

Slides 30 to 33

Let us understand how one-to-many and many-to-one relationships are implemented through JPA.

One-to-Many/Many-to-One Relationship 1-4

The diagram illustrates a one-to-many relationship. A green rounded rectangle labeled "Customer" is connected by a line to three yellow rounded rectangles, each labeled "Contact_number". This visualizes how one instance of the Customer entity can be linked to multiple instances of the Contact_number entity.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 30

One-to-Many/Many-to-One Relationship 2-4

- ❑ `javax.persistence.OnetoMany` annotation is used to set the relationship between two entities.
- ❑ `@OnetoMany` has the following attributes:
 - `targetEntity`
 - `cascade`
 - `fetch`
 - `mappedBy`
 - `orphanRemoval`
- ❑ To map the relationship between the two entities in the inverse, then Many-to-One associations can be built.
- ❑ `javax.persistence.ManyToOne` annotation is used to represent many-to-one relationships.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 31

One-to-Many/Many-to-One Relationship 3-4

- ❑ Following figure shows database schema of Item and Bid relationship in an auction:

The diagram shows the database schema for Item and Bid entities.
 - The Item entity is represented by a box containing fields: ITEM_ID (Primary Key), TITLE, DESCRIPTION, INITIAL_PRICE, BID_START_DATE, BID_END_DATE, and ITEM_SELLER_ID. An arrow labeled "Stored in" points from Item to a detailed table definition below it.
 - The Bid entity is represented by a box containing fields: BID_ID (Primary Key), AMOUNT, BID_DATE, BID_BIDDER_ID, and BID_ITEM_ID (Foreign Key). An arrow labeled "Stored in" points from Bid to a detailed table definition below it.
 - A double-headed arrow labeled "One-Many (Bidirectional)" connects the Item and Bid entities, indicating a bidirectional relationship between them.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 32

One-to-Many/Many-to-One Relationship 4-4

```

@Entity
@Table(name="ITEMS")
public class Item {
    @Id
    @Column(name="ITEM_ID")
    protected Long itemId;
    ...
    @OneToMany(mappedBy="item")
    protected Set<Bid> bids;
    ...
}

@Entity
@Table(name="BIDS")
public class Bid {
    @Id
    @Column(name="BID_ID")
    protected Long bidId;
    ...
    @ManyToOne
    @JoinColumn(name="BID_ITEM_ID",
                referencedColumnName="ITEM_ID")
    protected Item item;
    ...
}

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 33

Use slides 30 to 33 to explain the implementation of one-to-many and many-to-one relationship.

Explain them the scenario where a bank application keeps track of user's one primary contact number and multiple secondary contact numbers. In this case, each customer is associated with multiple contact number entities. The reference to the contact number entity is always through the Customer entity. Therefore, the Customer entity is the owning entity of the relationship and it is a unidirectional one-to-many relationship.

Then, explain the figure shown on slide 30 that presents the association of one **Customer** entity with many **Contact_Number** entities.

Discuss with them on how they will implement the relationship between the **Customer** and **Contact_Number** entities in the relational database.

Tell them that both the relations are implemented as primary key and foreign key association in the underlying database tables.

Then, tell them each record stored in the table is represented as entity object in the application. Discuss how they will hold multiple objects in the application retrieved from one-to-many relationship.

To hold the multiple object retrieved for the '**Many**' relationship, the collections types is used.

Use slide 31 to explain the mechanisms provided by JPA to implement one-to-many relationship. The **@OnetoMany** and **@ManytoOne** annotations can be defined on the field level or getter/setter methods of the property in the entity class.

The **javax.persistence.OnetoMany** annotation can be used to set the relationship between two entities. The attributes associated with **@OnetoMany** annotation are same as **OnetoOne** annotation; they are **targetEntity**, **cascade**, **fetch**, **mappedBy**, and **orphanRemoval**.

To map the relationship between the two entities in the inverse, Many-to-One associations can be built. The annotation **javax.persistence.ManytoOne** is used. The attributes associated with the **ManytoOne** annotation are **targetEntity**, **cascade**, and **fetch**.

Use slide 32 to demonstrate the example for one-to-many relationship. Explain the database schema shown in the figure shown on slide 32.

The @Many-to-one relationship is provided with @JoinColumn annotation. This is because the BID_ITEM_ID column is mapped as foreign key, so it is considered as the owing end of the relationship.

Use slide 33 to explain the code of one-to-many relationship. The code shows two different entities ITEMS and BIDS. Explain them that the mappedBy attribute of @One-to-Many annotation defines the one-to-many relationship. The @Many-to-One relationship is defined by @JoinColumn annotation. This is because the BID_ITEM_ID column is mapped as foreign key, so it is considered as the owing end of the relationship.

Slides 34 to 36

Let us understand many-to-many relationship.

Many-to-Many Relationship 1-3

The diagram illustrates a many-to-many unidirectional relationship. It features two orange rounded rectangles at the top labeled "Customer1" and "Customer2". From each customer, arrows point to three green rounded rectangles labeled "Contactno", "Contactno (Landphone)", and "Contactno". This indicates that multiple customers can be associated with multiple contact numbers, but the association is unidirectional from customers to contacts.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 34

Many-to-Many Relationship 2-3

The diagram illustrates a many-to-many bidirectional relationship. It shows two entities, "Category" and "Item", each represented by a rounded rectangle. A double-headed arrow between them is labeled "Many-to-Many (Bidirectional)". Below the entities, a "Join Table" is shown with two rows. The first row contains "CI_CATEGORY_ID (Foreign Key)" and "CI_ITEM_ID (Foreign Key)". The second row contains "CATEGORY_ID (Primary Key)" and "ITEM_ID (Primary Key)". This represents a many-to-many relationship where categories and items are connected through a join table.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 35

Many-to-Many Relationship 3-3

The diagram illustrates the implementation of a many-to-many relationship. It shows two entities, "Category" and "Item", each with its own table structure. The "Category" table has columns: "CATEGORY_ID (Primary Key)", "CATEGORY_NAME", "CREATION_DATE", and "MODIFICATION_DATE". The "Item" table has columns: "ITEM_ID (Primary Key)", "TITLE", "DESCRIPTION", "INITIAL_PRICE", "BID_START_DATE", "BID_END_DATE", and "ITEM_SELLER_ID". A "Join Table" is shown in the middle, labeled "Join Table", which contains "CI_CATEGORY_ID (Foreign Key)" and "CI_ITEM_ID (Foreign Key)". Arrows indicate that "Category" is stored in the "Category" table and "Item" is stored in the "Item" table.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 36

Use slides 34 to 36 to explain implementation of many-to-many relationship.

In a many-to-many relationship, each entity on either side of the relationship is associated with more than one entity on the other side. Many-to-many relationship can be a unidirectional or bidirectional relationship.

In a bank application, consider a scenario where multiple customers can provide the same landline phone number as a contact number and can also provide multiple contact numbers. Such an association is referred to as many-to-many relationship.

Many-to-Many unidirectional relationship has an owning entity through which the associated entity can be referenced but it is not the case in reverse direction. The figure shown on slide 34 shows a many-to-many relationship between the customer entity and the contact number entity.

Use slide 35 to explain many-to-many relationships.

In a many-to-many bidirectional relationship both the entities in the association can be referenced by more than one entity of the other type. In order to have a many-to-many relationship there needs to be a distinct join table that maps the many-to-many relationship. This is called an **association table**. Many application servers can automatically generate the association table for you. You need to specify the `@JoinTable` annotation to define the join table for Many-to-Many association. The `@JoinTable` annotation must be defined on both the sides of bi-directional relationship.

In a many-to-many bidirectional relationship, both the entities in the association can be referenced by more than one entity of the other type. JPA provides `javax.persistence.ManyToMany` annotation to implement many-to-many relationships.

Use slide 36 to explain many-to-many relationship. The slide shows the many-to-many relationship for a **Category** containing multiple Items and how an Item can belong to multiple **Category** entities. As shown in the figure on slide 36, the association or join table allows to indirectly match up primary keys from either side of the relationship by storing arbitrary pairs of foreign keys in a row.

Additional References:

For more information on the example of Many-to-Many relationship, you can refer the following links:

http://www.java2s.com/Tutorial/Java/0355__JPA/ManyToOneJoinTableJoinInverseJoinColumn.htm

<http://www.developerscrappad.com/139/java/java-ee/ejb3-jpa-entity-many-to-many-relationship/>

Slide 37

Let us understand entity inheritance.

Entity Inheritance

- Enterprise applications use inheritance to achieve code reuse and polymorphism.
- The inheritance in the application design has to be translated to the database design.
- JPA provides three different strategies for translating entity inheritance onto database:
 - A single table per class hierarchy
 - A table per concrete entity class
 - A table per sub class

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 37

Use slide 37 to explain entity inheritance. Object-oriented programs use entity inheritance to improve the efficiency of the application.

Discuss with them the main disadvantage of entity beans created in EJB2.0 was that they did not support inheritance and polymorphism. Entity beans provided an object-oriented view to the data persisted in the persistence storage. Thus, the features such as inheritance and polymorphism are key aspect to object-oriented programming paradigm.

Tell them that using inheritance may be appropriate when building groups of related beans that share common code.

Tell them that the JPA object-relational mapping engine provides support for inheritance hierarchies and polymorphic queries.

The inheritance in the object-oriented application has to be mapped onto programming code. JPA provides three strategies for translating entity inheritance:

- A single table per class hierarchy
- A table per concrete entity class
- A table per subclass

The default mapping strategy is single table per class hierarchy.

Slides 38 to 41

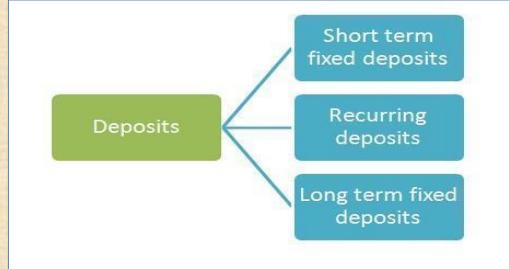
Let us understand single table per class hierarchy.

Single Table Per Class Hierarchy 1-4



All the entities in the class hierarchy are mapped onto a single table.

Consider the hierarchy as shown in the following figure:



Deposits

- Short term fixed deposits
- Recurring deposits
- Long term fixed deposits

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 38

Single Table Per Class Hierarchy 2-4



All the entities shown in the hierarchy are mapped onto a single table as shown in the given table.

Fixed deposit number	Linked savings Account number	Customer name	Amount
----------------------	-------------------------------	---------------	--------

The table definition requires a way for discriminating between the child classes of the `FixedDeposit` class.

Following is the table definition with discriminator column:

Fixed deposit number	Linked savings account number	Customer name	Amount	Fixed deposit type
----------------------	-------------------------------	---------------	--------	--------------------

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 39

Single Table Per Class Hierarchy 3-4



The discriminator column for the mapping strategy is defined through `javax.persistence.DiscriminatorColumn`.

Following are the attributes of the `DiscriminatorColumn`:

- name
- Discriminator type
- column definition
- length

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 40

Following code snippet demonstrates the single table per class hierarchy:

```

@Entity(name = "Account")
@DiscriminatorColumn(name = "DISCRIMINATOR",
discriminatorType = DiscriminatorType.STRING)
@DiscriminatorValue("account_type")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
class Account{
    @Id
    String account_type;
    @Id
    long account_number;
    ...
}

```

The discriminator is of type String. The property **account_type** of the entity is used to discriminate different types of deposits in the table.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 41

Use slides 38 to 41 to explain single table per class hierarchy.

This strategy is also referred to as Single-table strategy.

In this strategy, all the classes in the inheritance hierarchy are mapped as a part of single table. This means all the data of the superset as well as subset in the class hierarchy will be stored in the single table.

For instance, consider the hierarchy shown on slide 38. The hierarchy consists of four classes – Deposits, Short term deposits, Recurring deposits and Long term fixed deposits. The Deposits is the superclass in the class hierarchy which is also called as superset entity. The other three classes Short term deposits, Recurring deposits and Long term fixed deposits are inherited from the Deposits entity class. The instantiated objects of all these classes in the hierarchy are grouped into one table.

Different objects in the object-oriented class hierarchy are identified using a special column called a **discriminator column**. The discriminator column is used to implement the single-table strategy. It contains a value unique to the object type in a given row.

Use slide 39 to explain the table created according to the single table per class hierarchy strategy. The table definition requires a way for discriminating between the child classes of the Fixed Deposit class. Therefore, a discriminator column is added to the table as shown in the second table definition. The discriminator column is **Fixed Deposit Type**.

Then, using slide 40, explain the annotations used along with the single table per class hierarchy. The strategy of inheritance used is defined through `@InheritanceStrategy` annotation strategy. The `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)` annotation is used to provide the strategy type used.

JPA provides `javax.persistence.DiscriminatorColumn` annotation to define the discriminator column in the table.

The annotation has four attributes associated with it – name, Discriminator type, column definition, length.

- The name attribute represents the name of the column in the mapped table.
- The DiscriminatorType represents the data type of the discriminator column. The default type of the discriminator column is String. This column can assume String, char, or integer type.
- The columnDefinition attribute defines the method of generating the value in the column. This is generated by the persistence provider.
- The length attribute is used when the column type is String. The default value of the string length is 31.

Use slide 41 to explain the annotations used in the code of single table per class hierarchy.

The code on slide 41 shows the entity represented in the table through @Entity annotation. The @Discriminator annotation represents the name of the column of the table; it is defined as Discriminator in this case, which can be changed to type of account according to the context. This can also be specified through @DiscriminatorValue annotation as shown in the code.

Following code Snippet shows the example of using annotations to design single-table inheritance hierarchy for a module named **Account** and its types **SavingAccount** and **CurrentAccount**. The primary table is named as **Bank_Account** where different types of Accounts are mapped.

```
// This is the base class in Single Table Inheritance
@Entity
@Table(name="Bank_Account")
@Inheritance(strategy=SINGLE_TABLE, discriminatorValue="acc")
@DiscriminatorColumn(name="Account_TYPE")
public class Account implements Serializable {
    ...
}

@Entity
@Inheritance(discriminatorValue="saving")
public class SavingAccount extends Account {
    ...
}

@Entity
@Inheritance(discriminatorValue="current")
public class CurrentAccount extends Account {
    ...
}
```

Note, that the use of @Inheritance annotation is not needed in derived classes SavingAccount and CurrentAccount, respectively.

Tips:

The SINGLE_TABLE mapping strategy is the simplest to implement and performs better than all the inheritance strategies. The JPA engine does not generate any complex joins, unions, or subselects when loading the entity or when traversing a polymorphic relationship, because all data is stored in one table. However, the biggest issue with this strategy is that all columns of subclass properties must be nullable and results in columns with null values. Thus, this strategy does not support the features of normalization very well.

In-Class Question:

After you finish explaining single table per class hierarchy, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which column is required to implement Single-table inheritance hierarchy?

Answer:

Discriminator column

Slides 42 and 43

Let us understand the table per concrete entity class strategy.

Table Per Concrete Entity Class Strategy 1-2



❑ A table is created for every entity class in the hierarchy.

- This strategy corresponds to **InheritanceType.TABLE_PER_CLASS**.

❑ Each of these tables has columns which are properties of the sub class.

❑ This strategy does not require a discriminator column as in the case of single table per class hierarchy.

❑ In order to extract data from individual classes:

- **A separate query is written on the individual tables or SQL UNION queries are used.**



© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 42

Table Per Concrete Entity Class Strategy 2-2



❑ Following figure shows the class design and database table design for the Table per Concrete class:

Class Diagram

Database Table Design

Powered by  Enterprise Application Development in Java EE/Session 10 43

Use slides 42 and 43 to explain the table per concrete entity class strategy.

Tell them that the table per concrete strategy is also referred to as **Table-per-class** strategy.

According to this strategy, a separate table is created for each concrete entity in the inheritance hierarchy. The attributes of each table are the properties of the subclass. Since, different types of entities are not merged, this strategy does not require a discriminator column.

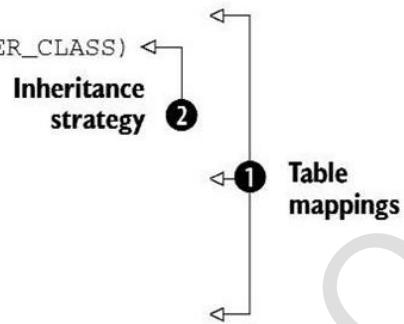
For instance, the example of **Deposits** discussed for table per class hierarchy, there are three concrete entity classes – Short term fixed deposits, Recurring deposits and Long term fixed deposits. Three tables are created on the database. Each of these tables has columns which are properties of the subclass.

Following figure shows the inheritance mapping using the table-per-class strategy:

```

@Entity
@Table(name="USERS")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class User {
...
@Entity
@Table(name="SELLERS")
public class Seller extends User {
...
@Entity
@Table(name="BIDDERS")
public class Bidder extends User {

```



The inheritance strategy

`@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)` which is specified on the parent or superclass. However, all the other concrete classes or subclasses are mapped to separate tables using `@Table` annotation.

In order to extract data from individual classes, a separate query is written on the individual tables or SQL UNION queries are used. This strategy provides poor support for polymorphic relationships.

Slide 43 displays the table design and class diagram mapping for table per concrete class hierarchy. The example shows a hierarchy where the class Payment is the super class with subclasses Online Transfer, Cheque payment and Cash payment.

Each of the classes is mapped onto a separate table, even if they are inherited from the superclass. Each class is having its own primary key PAYMENT_ID_PK.

Slides 44 and 45

Let us understand table per subclass strategy.

Table Per SubClass Strategy 1-2

- Is also known as joined class strategy.
- Annotated with `javax.persistence.Joined`.
- The super class or the root of the class hierarchy is represented by a single table.
- Each subclass of the hierarchy is mapped as a different table.
 - Columns of these tables are the properties of the subclass.
- Each sub class has its own primary key.
- All the entities of the hierarchy can be aggregated by applying a **JOIN** operation on all the subclasses.
- This strategy provides good support for polymorphic relationships.




© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 44

Table Per SubClass Strategy 2-2

- The hierarchy can be implemented using the following tables:

FixedDeposit		
Fixed Deposit number	LinkedAccount number	Amount

Short Term Fixed Deposits		
Short term FD number	Linked savings account number	Amount

Recurring Deposits		
Recurring Deposit number	Linked savings account number	Amount

Long Term Fixed Deposits		
Long term FD number	Linked savings account number	Account



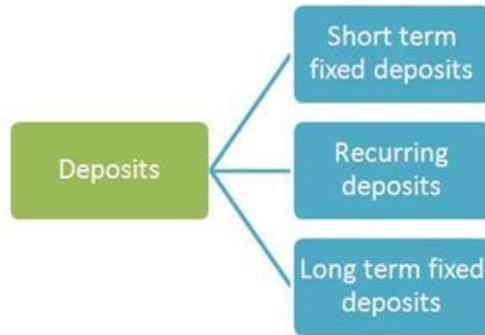

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 45

Use slides 44 and 45 to explain table per subclass strategy.

This strategy is also known as Joined class strategy. This strategy corresponds to `InheritanceType.JOINED`.

The joined-tables inheritance strategy uses one-to-one relationships to model object-oriented inheritance. It involves creating separate tables for each entity and relating direct descendants in the hierarchy with one-to-one relationships.

Consider the following inheritance discussed in the table per class strategy:



Each subclass in this hierarchy is mapped onto a different table. There are entries of all deposits in the superclass table. The superclass and the subclass are related through primary key-foreign key relationship.

Following code Snippet shows the implementation of Joined class strategy with **Account** and its subclasses **SavingAccount** and **CurrentAccount**:

```

// This is the base class in Joined Class Inheritance
@Entity
@Table(name="Bank_Account")
@Inheritance(strategy=JOINED, discriminatorValue="acc")
@DiscriminatorColumn(name="Account_TYPE")
public abstract class Account implements Serializable {
    ...
}

// This is derived class in joined subclass hierarchy
@Entity
@Table(name="Saving_Account")
@Inheritance(discriminatorValue="saving")
@PrimaryKeyJoinColumn(name="Account_ID")
public class SavingAccount extends Account {
    ...
}

// This is derived class in joined subclass hierarchy
@Entity
@Table(name="Current_Account")
@Inheritance(discriminatorValue="current")
@PrimaryKeyJoinColumn(name="Account_ID")
public class CurrentAccount extends Account {
    ...
}
  
```

Note, that the **@DiscriminatorColumn** and **@DiscriminatorValue** annotations in exactly the same way as the single-table strategy. However, the strategy element is specified as JOINED,

The one-to-one relationship between parent and child tables will be implemented through the `@PrimaryKeyJoinColumn` annotations in both the `SavingAccount` and `CurrentAccount` entities. In both cases, the `name` element specifies the `Account_ID` foreign key.

The table per subclass strategy supports polymorphic queries, however, it requires joining of multiple tables. When instantiating entity subclasses, it may result in poor performance in case of extensive class hierarchies.

In-Class Question:

After you finish explaining the table per subclass strategy, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which relationship is used by table per subclass hierarchy to relate the superclass and the subclasses?

Answer:

Primary key-foreign key relationship

Slide 46

Let us understand the annotations associated with entity inheritance mapping.

Annotations Used For Entity Inheritance Mapping

- ❑ The hierarchy in an application can be identified while deploying by annotating the root class of the hierarchy with `javax.persistence.Inheritance`.
- ❑ The mapping strategy is defined through the annotation `javax.persistence.InheritanceType`.
- ❑ The `InheritanceType` annotation can assume any one of the following values:
 - `SINGLE_TABLE` (default) corresponds to single table per class hierarchy.
 - `JOINED` value corresponds to a table per subclass strategy.
 - `TABLE_PER_CLASS` value corresponds to a table per concrete entity class strategy.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 46

Use slide 46 to explain the annotations used for entity inheritance mapping onto the database.

As already discussed in the earlier slides, brief the annotations used for mapping the three strategies provided by JPA.

The mapping strategy of a hierarchy can be defined by annotating the root class of the hierarchy with `javax.persistence.Inheritance`. This annotation also defines the strategy of mapping the inheritance onto the database. The strategy of mapping the inheritance onto the database is defined through the annotation `javax.persistence.InheritanceType`. This annotation can assume any one of the three values – `SINGLE_TABLE`, `JOINED`, and `TABLE_PER_CLASS`. The default inheritance mapping strategy is `SINGLE_TABLE`.

Additional References:

You can get more examples to prepare the inheritance to be taken in the class. You can refer this link to prepare the demonstrations: <http://what-when-how.com/enterprise-javabeans-3-1/entity-inheritance-enterprise-javabeans-3-1/>

Slide 47

Let us understand non-entity base classes.

Non-Entity Base Classes

□ Entity classes can also be inherited from non-entity base classes.

Abstract Entity classes

- Cannot be instantiated.
- Concrete entity classes can extend and define the functionality of these abstract entity classes.
- Prefixed with 'abstract' keyword.

Mapped super classes

- Are those classes in the enterprise application which are not persisted.
- Applications may inherit the behavior and properties of such super class but the state of the mapped super classes are not persisted.
- Annotated with @MappedSuperClass.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 47

Use slide 47 to explain non-entity base classes. Non-entity base classes correspond to those classes in the hierarchy which cannot be instantiated. There are two types of non-entity base classes – Abstract Entity classes and Mapped Super classes.

Abstract entity classes are similar to concrete entity classes, but they cannot be instantiated. Concrete entity classes can extend and define the functionality of these abstract entity classes. Whenever a query is executed on abstract entity classes, the query is executed on all the concrete subclasses of the abstract entity class. Abstract entity classes are prefixed with 'abstract' keyword.

Mapped Super classes are those classes in the enterprise application which are not persisted. They are not annotated with @Entity annotation, but they are mapped with @MappedSuperclass annotation.

In-Class Question:

After you finish explaining non-entity base classes, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which of the following strategy creates the normalized schema for the inheritance hierarchy?

Answer:

Table per subclass strategy

Additional References:

To get examples on non-entity classes, you can refer the following link:

<http://docs.oracle.com/javaee/6/tutorial/doc/bnbqn.html>

Slide 48

Let us summarize the session.

Summary

- ❑ Relationships describe how objects collaborate with one another, to contribute to the behavior of the system.
- ❑ A database schema describes the table structure, data types, and relations in a database.
- ❑ Some of the common terms in relational database are attributes, primary key, and foreign key.
- ❑ There are four types of relationships that can be created between the entities in the database that includes one-to-one, one-to-many, many-to-one, and many-to-many.
- ❑ Entity beans represent objects in OOAD.
- ❑ In order to support entity relationships, the object-to-relational mapping engine must provide support for object-oriented features such as inheritance, polymorphism, and so on.
- ❑ Enterprise applications use inheritance among entities for code reuse and to implement polymorphism.
- ❑ Mapping of persistent application objects onto the database can be defined through annotations in JPA.
- ❑ The association among entities is defined through relationships that can be unidirectional or bidirectional.
- ❑ JPA defines three strategies for mapping the entity inheritance onto the database.
- ❑ javax.persistence.Inheritance and javax.persistence.InheritanceType are the annotations used to map inheritance onto the database.
- ❑ Abstract Entity classes and Mapped super classes are non-entity base classes.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 10 48

Using slide 48, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

10.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concepts of Query and Criteria API that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 11 – Query and Criteria API

11.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

11.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain Java Persistence Query Language (JPQL)
- List the characteristics of JPQL
- Explain Query and TypedQuery API
- Explain how to write the named queries in JPQL
- Explain how to execute the named queries in JPQL
- Explain the methods to tune the JPQL query results
- Explain polymorphic queries
- Explain Criteria API
- Understand how to develop queries using Criteria API
- Explain how to manage Criteria query results
- Explain Metamodel API

11.1.2 Teaching Skills

To teach this session successfully, you should be aware of Java Persistence Query Language (JPQL). JPQL is used in applications to write SQL like queries on application databases. JPQL is provided by JPA to manipulate databases from object-oriented code in the application. You should be aware of how to write queries using Query and TypedQuery classes and have knowledge of writing named queries in the application. You should also be aware of how to tune performance of the queries and be able to write polymorphic queries.

You should prepare yourself with Criteria API which is an alternative method of writing queries in object-oriented way. You should be aware of how to manage Criteria query results and work with Metamodel API.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

Objectives

- Explain Java Persistence Query Language (JPQL)
- List the characteristics of JPQL
- Explain Query and TypedQuery API
- Explain how to write the named queries in JPQL
- Explain how to execute the named queries in JPQL
- Explain the methods to tune the JPQL query results
- Explain polymorphic queries
- Explain Criteria API
- Understand how to develop queries using Criteria API
- Explain how to manage Criteria query results
- Explain Metamodel API

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 2

Tell them that they will be introduced to Java Persistence Query Language (JPQL) provided by JPA. They will learn to write database related queries using Query and TypedQuery classes provided by Java, static and dynamic queries and also execute polymorphic queries on the database of the application.

Tell them that they will also learn about Criteria API and write Criteria queries to be used in Java programs, the Metamodel API and its utility in the application.

11.2 In-Class Explanations

Slide 3

Let us understand the definition of JPQL.

Introduction

- ❑ Querying is the process of extracting data from the database.
- ❑ Structured Query Language (SQL) is used to retrieve data from the databases.
- ❑ Java Persistence Query Language (JPQL) is equivalent to SQL and is used by JPA.
- ❑ **JPQL:**
 - Is a string-based query language which allows the developers to write queries over entity objects created in the enterprise applications.
 - Offers queries that follow object-oriented approach.
 - Makes the application independent of a particular database schema.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 3

Use slide 3 to explain what JPQL is. JPQL is a query language similar to Structured Query Language (SQL). SQL is used in relational databases to query and retrieve data from the database. JPQL queries are written in a way compatible with rest of the code in a Java application.

JPQL is a string based query language, where developers can write queries to be executed on relational database of the application. The main benefit of JPQL is that it allows the developers to write the queries in portable format. The JPQL queries follow object-oriented approach and make the application independent of a particular database schema.

Slide 4

Let us understand the characteristics of JPQL.

Characteristics of JPQL

- ❑ Some of the characteristics of JPQL are as follows:
 - Is a query specification language.
 - Enables writing static and dynamic queries on persisted entities.
 - Compiles to a target language such as SQL.
 - Allows the developer to write queries on Abstract Persistence Schema.
 - Uses SQL like syntax to retrieve objects or values based on abstract schema.
 - Allows defining queries in meta data annotations or deployment descriptor.

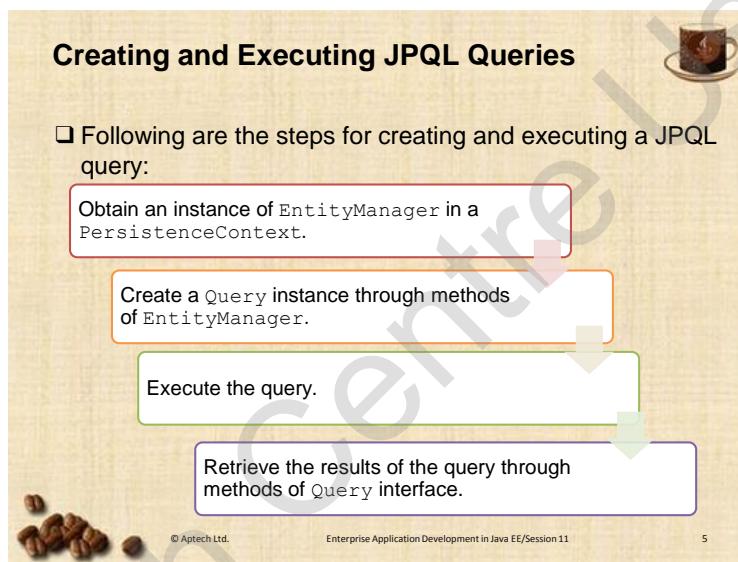
© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 4

Use slide 4 to explain the characteristics of JPQL.

- It is a query specification language for creating static and dynamic queries on persisted entities.
- It is defined by JPA specification.
- It is an object-based query language.
- It allows the developers to write queries over stored entities, their state, and relationships.
- It can be compiled to a target language, such as SQL understood by relational databases.
- It allows writing queries on the abstract persistence schema which contains entities and their relationships. The queries written on abstract persistence schema are translated to the specific database schema by JPA.
- It uses SQL-like syntax to retrieve objects or values based on the abstract schema.
- It allows defining the queries in metadata annotations or in the XML descriptor.

Slide 5

Let us understand the steps involved in executing a JPQL query.



Use slide 5 to explain the steps involved in creating and executing a JPQL query.

A JPQL query is created through an instance of EntityManager. An EntityManager instance is associated with a PersistenceContext of the application. It manages all the active entities in the application.

A Query instance is created through the `createQuery()` method of EntityManager class.

Create a Statement object with the query string as parameter and execute the query through `executeQuery()` or `execute()` methods of Statement class. These methods will return ResultSet objects.

The results can then be retrieved and managed through methods of Query interface.

Slides 6 and 7

Let us understand named queries.

JPQL Named Query 1-2



- Defines a predefined static query string.
- Are created when certain query has to be executed repeatedly.
- Created through `javax.persistence.NamedQuery` annotation that has four attributes namely, name, query, hints, and lockMode.

```
@NamedQuery(name = "Account.All"
query = "Select a from Account a")
public class Account
implements Serializable {
    ...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 6

JPQL Named Query 2-2



Following table lists the attributes and descriptions of the `@NamedQuery` annotation:

Description	Attribute
Refers to the query.	Name
Refers to the query string written in the JPQL text format.	Query
Refers to the hints and properties related with the query.	Hints
Used in query execution.	LockMode

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 7

Use slides 6 and 7 to explain named queries.

Named queries are used to execute static queries in the application. They are defined through `@javax.persistence.NamedQuery` annotation. The `@NamedQuery` annotation has four attributes – name, query, hints, and lockmode.

The Name attribute is used to identify the query. It is similar to variable name a string query is associated with this name. The query attribute represents the SQL string which has to be executed. hints attribute can be used to define the properties of the query. lockmode attribute defines the types of locks that have to be acquired to execute the named query.

The code on slide 6 shows a named query which retrieves all the data from the table Account.

Slide 8

Let us understand how to define multiple named queries in an application bean.

Multiple Named Queries

- ❑ Developers can attach multiple named queries to the same entity class through `@NamedQueries` annotation.
- ❑ Following code snippet shows the usage of `@NamedQueries` annotation:

```
@NamedQueries({
    @NamedQuery(name="Account.findAll",
        query="SELECT a FROM Account a"),
    @NamedQuery(name="Account.findByName",
        query="SELECT a FROM Account a WHERE
            a.name = :name")
})
public class Account implements Serializable {
    ...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 8

Use slide 8 to explain how to execute multiple named queries. JPA provides `@NamedQueries` annotation which allows multiple named queries to be incorporated in it.

The `@NamedQueries` annotation incorporates `@NamedQuery` annotations where each `@NamedQuery` annotation represents a JPQL query to be executed.

Additional References:

To get additional example on writing named queries, refer to the following links:

<http://www.java2s.com/Code/Java/JPA/CreateMorethanoneNamedQueriesforanEntity.htm>.



Where can you apply the `@NamedQueries` annotation?

Answer:

The `@NamedQueries` annotation can be applied to an entity or mapped superclass.

Slide 9

Let us understand how to execute dynamic JPQL queries.

JPQL Dynamic Query

- ❑ Dynamic queries are used when the application has to build queries at runtime.
- ❑ Created through the `createQuery()` method of the `EntityManager` interface.
- ❑ Following code snippet demonstrates creation of dynamic queries:

```
@Stateless
public class AccountStatelessSessionBean implements AccountBean {
    ...
    @PersistenceContext
    public EntityManager e;
    public List displayAccounts() {
        Query Q = e.createQuery("select a from Account a", Account.class);
        List accountsList = Q.getResultList();
        ...
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 9

Use slide 9 to demonstrate how developers can create dynamic JPQL queries.

Dynamic queries are used when the application have to build queries at run time. They can be created based on certain conditions.

Dynamic queries can be created using `createQuery()` method of `EntityManager` class.

The code explained on slide 9 shows how a `Query` object can be created through `createQuery()` method in a `PersistenceContext`.

In the code snippet, a query has been created with a criteria or a condition provided through ‘where’ clause. The query retrieves the `Account` object for all the customers and returns as a list of `Account` objects from the `displayAccounts()` method.

Additional References:

You can get the additional example to implement JPQL dynamic query on the following link:
https://docs.oracle.com/html/E13981_01/ent30qry002.htm.

Slide 10

Let us understand the `Query` interface.

Query Interface 1-2

- ❑ `Query` interface provides various methods to:
 - set parameters to the query.
 - execute the queries.
 - retrieve the results of the query.
 - set the pagination properties of the result set.
 - control the flush mode.
- ❑ JPA 2.0 introduces `TypedQuery` interface which extends the `Query` interface.
 - It is usually used when you expect the query to return more specific result types

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 10

Use slide 10 to introduce the `Query` interface.

`Query` interface provides various methods to execute the queries. The `Query` interface provides various methods to set the parameters of the query, to execute the queries, and to retrieve results of the query. The `Query` interface also provides methods to set the pagination properties of the result set and control the flush mode. It is mainly used when the result type is unknown or when a query returns polymorphic results.

`TypedQuery` extends the `Query` class and controls the execution of query. `TypedQuery` interface is used when the developer knows the type of result that the query will return.

Slide 11

Let us understand the methods provided by the Query interface.

Query Interface 2-2

- ❑ Following table shows methods provided by `Query` interface for query execution:

Method	Description
<code>List getResultList()</code>	Used to extract the result set of a query.
<code>Object getSingleResult()</code>	Used to extract one object from the result set.
<code>int executeUpdate()</code>	Used to execute an update or delete statement on the database.
<code>Query setMaxResult()</code>	Used to set the maximum number of results that can be received from a query.
<code>Query setFirstResult()</code>	Used to set the position of the first result the query is set to receive.
<code>void setParameter()</code>	Is an overloaded method and is used to set parameters for query execution.
<code>void setFlushMode()</code>	Used to set the flush mode to execute the query.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 11

Use slide 11 to explain the methods provided by the `Query` interface.

Slide 12

Let us understand the execution process of named queries.

Executing the Named Query

- ❑ Following code snippet demonstrates the execution of named queries:

```
@Stateless
public class AccountStatelessSessionBean implements AccountBean {
    @PersistenceContext(unitName="mypersistenceunit")
    private EntityManager e;
    public List findAccounts() {
        Query q = e.createNamedQuery("Account.findAll", Account.class);
        List acclist = q.getResultList();
        return acclist;
    }
}
```

- ❑ The `findAccounts()` method invokes the `createNamedQuery()` method on the entity manager, which returns the `Query` object.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 12

Use slide 12 to explain the execution of named queries in the application.

Using slide 8, creating a named query on the `Account` entity class was demonstrated. The named query was labelled `findAll`, the query here is created from the named query on `Account` entity class.

The result of the query is retrieved into a `List` object `acclist`.

In the code snippet, an instance of EntityManager is injected in the AccountStatelessSessionBean class. The `findAccounts()` method invokes the `createNamedQuery()` method on the entity manager, which returns the `Query` object. The `Query` object contains the list of all Accounts retrieved from the Account entity result after execution of the query is retrieved in a list of strings. Then, the retrieved Account objects list is returned from the method to the calling environment.

Slide 13

Let us understand parameterized queries.

Parameterized Queries

- ❑ JPQL allows execution of parameterized queries.
- ❑ Parameters are dynamically passed during execution for these queries.
- ❑ The `setParameter()` method of `Query` interface is used to set parameters.
- ❑ The parameters are passed in two ways:
 - Based on the name of the parameters
 - Based on the position of the parameters

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 13

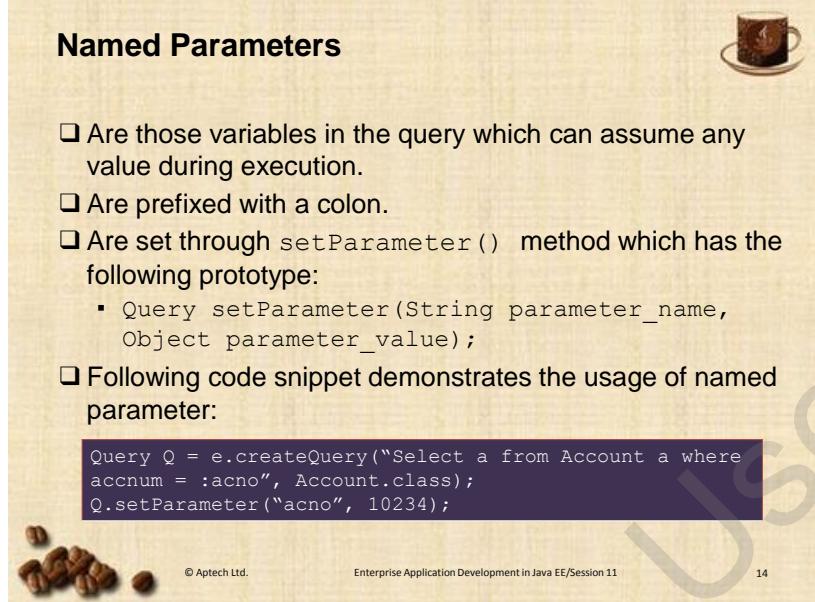
Use slide 13 to explain parameterized queries.

Parameterized queries are queries to which parameters can be explicitly passed. This query can be executed multiple times with different values as parameters.

The parameters of the query are dynamically passed during application execution. Applications use `setParameter()` method to define the parameters of the query. Parameters can be passed by name of the parameter or by referring to the positions of the variables in the query.

Slide 14

Let us understand how parameters can be defined through names.



Named Parameters

- ❑ Are those variables in the query which can assume any value during execution.
- ❑ Are prefixed with a colon.
- ❑ Are set through `setParameter()` method which has the following prototype:
 - `Query setParameter(String parameter_name, Object parameter_value);`
- ❑ Following code snippet demonstrates the usage of named parameter:


```
Query Q = e.createQuery("Select a from Account a where
accnum = :acno", Account.class);
Q.setParameter("acno", 10234);
```

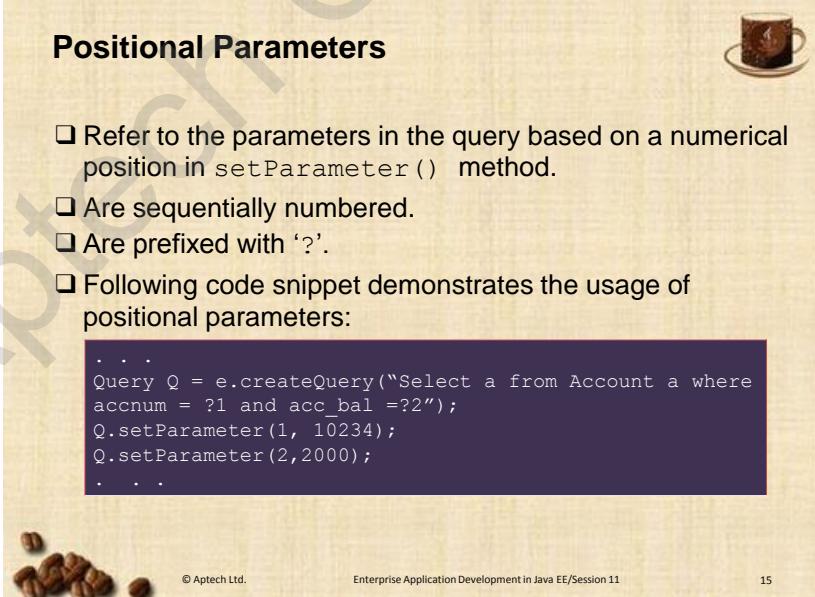
© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 14

Use slide 14 to explain the usage of parameters in code.

Named parameters can assume any values in a query and are prefixed by a colon. The values of the named parameters can be set through `setParameter()` method. Then, demonstrate the syntax of using the Named parameter and its usage as shown on slide 14.

Slide 15

Let us understand positional parameters.



Positional Parameters

- ❑ Refer to the parameters in the query based on a numerical position in `setParameter()` method.
- ❑ Are sequentially numbered.
- ❑ Are prefixed with ‘?’.
- ❑ Following code snippet demonstrates the usage of positional parameters:


```
...
Query Q = e.createQuery("Select a from Account a where
accnum = ?1 and acc_bal =?2");
Q.setParameter(1, 10234);
Q.setParameter(2,2000);
..."
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 15

Use slide 15 to explain positional parameters in named queries. The values of the parameters can be set based on the value of its occurrence in the query.

The syntax for assigning the positions is to prefix a ‘?’ to the position. These numbers are used to pass values for each of the parameters in the query. The `setParameter()` method is used to set the positional parameters.

In the given example, the query requires two parameters for execution. These parameters numbered 1 and 2 are prefixed with a ‘?’ . While setting the parameter values with `setParameter()` method, these positions are used to set the values.

Slide 16

Let us understand temporal parameters.

Temporal Parameters

- ❑ Temporal parameters are also referred to as Date parameters.
- ❑ Following are the types of objects which are passed as temporal parameters:
 - Date
 - Time
 - TimeStamp
- ❑ Temporal parameters are passed through the following `setParameter()` method:

```
query.setParameter("date", new java.util.Date(),
TemporalType.DATE);
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 16

Use slide 16 to explain temporal parameters in queries. Temporal parameters are those whose data type is Date, Time or TimeStamp.

Passing date parameters to the query is different from passing basic data type parameters to the queries. Date parameters passed to the JPQL queries are objects of Date, Time, and TimeStamp classes. These parameters are collectively referred as temporal parameters. The Query interface provides various definitions of `setParameter()` methods to use temporal parameters in queries.

Explain the syntax of the `setParameter()` method used for the purpose. `setParameter()` method is an overloaded method.

Slide 17

Let us understand how the performance of the queries can be tuned to improve.

Tuning the Queries

- ❑ Query interface provides various tuning methods that are used to effect the result of the query execution.
 - `getResultSet()`
 - `getSingleResult()`
- ❑ These methods can be invoked before the query execution to set the result.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 17

Use slide 17 to explain the methods provided by Query interface for performance tuning of the query.

Query interface provides `getResultSet()` and `getSingleResult()` methods to retrieve the result of an executed query.

The `getResultSet()` method retrieves the output of the query as a list and `getSingleResult()` method retrieves only one value of the query.

It is not preferable to retrieve all the results of an executed query, retrieving excessive results than those required would deteriorate the performance of the application as it uses excessive memory than required. Query interface provides methods which avoid retrieving excessive number of results against those required.

In-Class Question:

After you finish explaining the performance of Query interface, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which of the following method is used to retrieves the output of the query as a list?

Answer:

`getResultSet()` method.

Slide 18

Let us understand methods used for paging the result of a query executed.

Paging the Result

- ❑ Paging is the process of accessing a subset of results of the query.
- ❑ Following are the methods used for paging the result of a query:
 - `getMaxResults()` – is used limit the maximum number of results from query execution.
 - `setFirstResult()` – is used for processing the output of an executed query.
- ❑ Following code snippet shows the usage of `getMaxResults()` and `setFirstResult()` methods:

```
...  
Query Q = e.createQuery("Select a from Account");  
Q.setParameter("acno", 10234);  
List results =  
Q.setFirstResult('10234').setMaxResult(1).getResultList();  
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 18

Use slide 18 to explain the methods used to limit the number of results retrieved as the result of a query executed or retrieve a subset of the results of a query.

The `getMaxResults()` is used to limit the maximum number of results from query execution. The method `setFirstResult()` is used for processing the output of an executed query. This method is used to set the initial value in the result set from where the output can be displayed.

Demonstrate the usage of `setMaxResult()` and `setFirstResult()` methods.

In the scenario, the first value of the result set is set to the value 'Alex'. The `setMaxResult()` method defines the maximum number of values that can be retrieved as a result of the query, in this case it is set to 1. The code also sets the first result to the row with account number 10234. Based on these settings the result for this query is retrieved by the `getResultList()` method.

Slide 19

Let us understand Query Hints.

Query Hints

- ❑ Hints are set along with the queries to define additional information which can be used while executing a query.
- ❑ The `setHint()` method is used to define hints which has the following syntax:

```
Query setHint(String hint_name, Object value);
```

- ❑ Following code snippet demonstrates the usage of `setHint()` method:

```
...
Query Q = e.createQuery("Select Customer_name from
BankAccount where accnum = :acno");
Q.setParameter("acno", 10234);
Q.setHint("timeout", 1000);
...
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 19

Use slide 19 to explain query hints.

Developers use hints to define additional information used for executing the query such as defining a time out for the query execution.

Query class provides `setHint()` method to provide this additional information.

Demonstrate the syntax and usage of `setHint()` method using given slide 19. The given code sets the timeout period for query execution as 1000 milliseconds.

In-Class Question:

After you finish explaining the Query hints, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the use of `setHint()` method?

Answer:

Defines additional information used for executing the query.

Slide 20

Let us understand flush mode.

Flush Mode

- ❑ When an application performs an operation on the database, the changes are synchronized with the database.
- ❑ The EntityManager invokes `flush()` method to synchronize the changes with the database.
- ❑ There are two flush modes – `AUTO` and `COMMIT`.
- ❑ The `setFlushMode()` method is used to set the flush mode.
- ❑ Following is the syntax of `setFlushMode()` method:
`Query.setFlushMode(FlushModeType flushmode);`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 20

Use slide 20 to explain flush mode. Flush mode determines when the changes made to the application are persisted onto the database.

The EntityManager of the PersistenceContext determines the flush mode of the database. There are two flush modes – `AUTO` and `COMMIT`.

In case of `AUTO` flush mode the `flush()` method is invoked before executing a correlated query.

In case of `COMMIT` flush mode the `flush()` method is invoked when a transaction is committed.

The flush mode is set through `setFlushMode()` method of the EntityManager.

In-Class Question:

After you finish explaining the flush mode, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



When the `flush()` method is invoked in case of `COMMIT`?

Answer:

The `flush()` method is invoked when a transaction is committed.

Slide 21

Let us understand the clauses which can be used with WHERE clause.

WHERE Clause

- ❑ Used to specify a condition on the result set.
- ❑ Can apply multiple conditions using AND and OR operators.
- ❑ Following are the clauses which can be used with the WHERE clause:

DISTINCT	IN	LIKE
BETWEEN	IS NULL	IS EMPTY

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 21

Use slide 21 to explain WHERE clause used in JPQL queries. SQL queries use WHERE clause to specify conditions based on which data can be retrieved. 'WHERE' clause in JPQL queries has the same function, it allows for usage of additional clauses as shown on slide 21 so that developers can add additional conditions.

DISTINCT clause is used to retrieve unique results from the query.

IN clause is used to select a value which is in a set of values.

LIKE clause is used to select a value based on pattern matching.

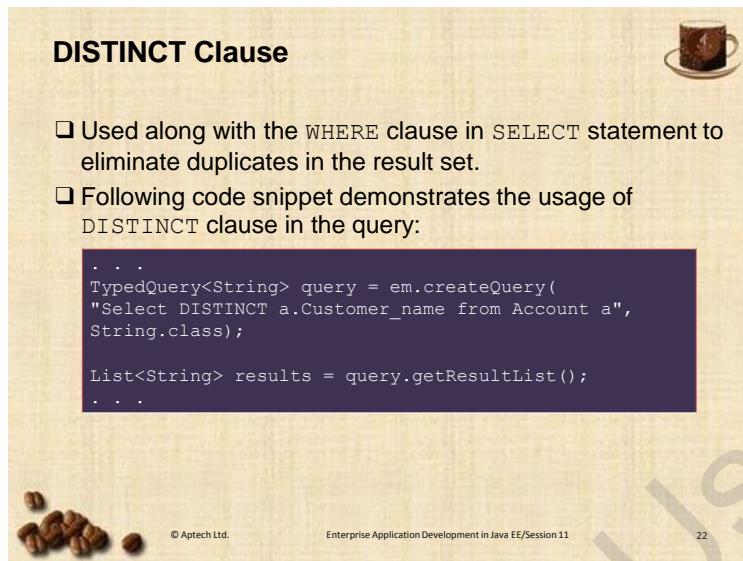
BETWEEN clause allows the developer to provide a range of values from which the results can be selected.

IS NULL value used to check whether the values selected have any NULL values.

IS EMPTY clause is used to check whether the result set is empty.

Slide 22

Let us understand the usage of DISTINCT clause with the help of an example.



DISTINCT Clause

- Used along with the WHERE clause in SELECT statement to eliminate duplicates in the result set.
- Following code snippet demonstrates the usage of DISTINCT clause in the query:

```

...
TypedQuery<String> query = em.createQuery(
    "Select DISTINCT a.Customer_name from Account a",
    String.class);

List<String> results = query.getResultList();
...

```

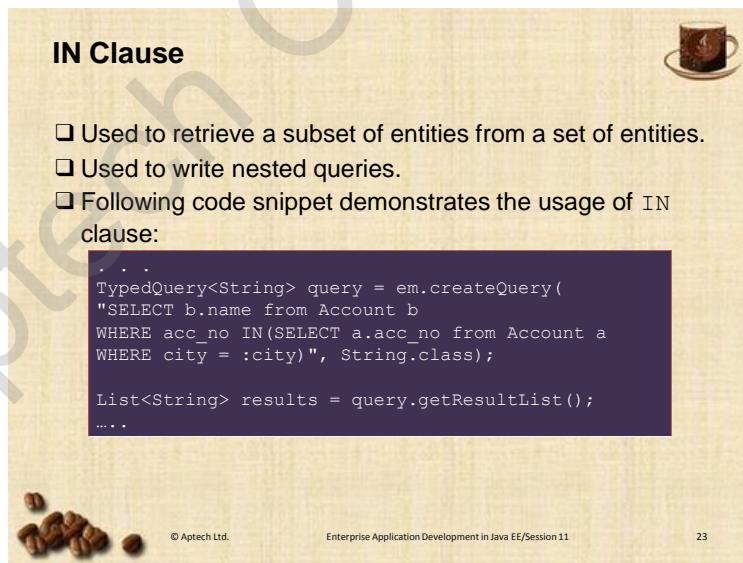
© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 22

Use slide 22 to explain the usage of DISTINCT clause in a JPQL query. The query defined in the `createQuery()` method selects distinct names of the customers who hold an account.

In the code snippet, the `createQuery()` method is used to create the query and `getResultList()` method of the `Query` interface to retrieve the executed results. The query retrieves the set of unique customer names from the `Account` entities.

Slide 23

Let us understand the usage of IN clause.



IN Clause

- Used to retrieve a subset of entities from a set of entities.
- Used to write nested queries.
- Following code snippet demonstrates the usage of IN clause:

```

...
TypedQuery<String> query = em.createQuery(
    "SELECT b.name from Account b
    WHERE acc_no IN(SELECT a.acc_no from Account a
    WHERE city = :city)", String.class);

List<String> results = query.getResultList();
...

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 23

Use slide 23 to explain the usage of IN clause.

The query defined in the `createQuery()` method is a nested query. The inner query selects the account numbers held in a particular city, the value of city is accepted as a parameter from the user. The outer query further selects the name of the account holders who hold these accounts.

Slide 24

Let us understand the usage of LIKE clause.

LIKE Clause

- ❑ Used to match patterns in the query.
- ❑ Following code snippet demonstrates the usage of the LIKE clause:

```
. . .
TypedQuery<String> query = em.createQuery(
    "SELECT AVG(acc_bal) from Account a
     WHERE city LIKE 'B%'"
    GROUP BY city", String.class);
List<String> results = query.getResultList();
. . .
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 24

Use slide 24 to explain the usage of LIKE clause in the query.

The query defined in the `createQuery()` method retrieves the average balance of all the accounts held in a city starting with B.

Slide 25

Let us understand the BETWEEN clause.

BETWEEN Clause

- ❑ Used to specify a range of a property value in the WHERE clause.
- ❑ Following code snippet demonstrates the usage of BETWEEN clause:

```
. . .
TypedQuery<String> query = em.createQuery("SELECT
a.acc_no, a.name FROM Account a
WHERE a.acc_bal BETWEEN :min_sal and :max_sal"
, String.class);
List<String> results = query.getResultList();
. . .
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 25

Use slide 25 to explain the usage of BETWEEN clause in the application.

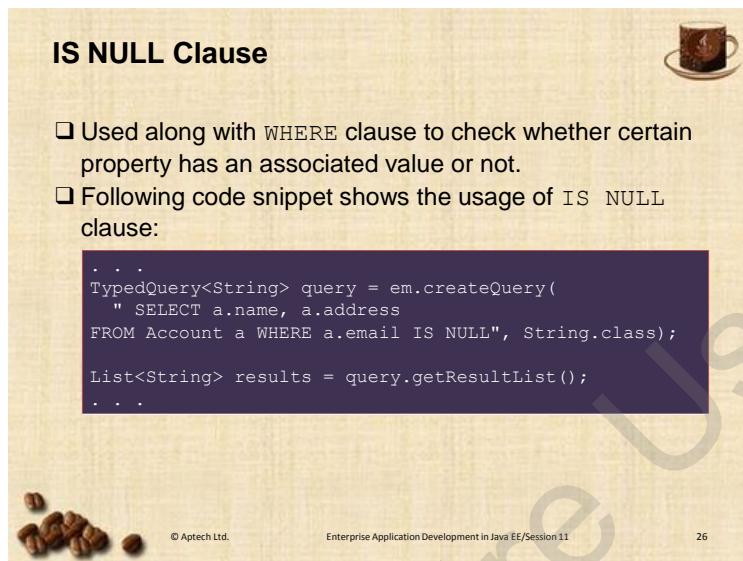
The BETWEEN clause is used to retrieve results which lie in a given range of values.

The query defined in the `createQuery()` method has two variables `min_sal` and `max_sal`. The query retrieves the account number and corresponding customer name, whose account balance lies between this range.

The values of these parameters are to be provided at run time. The select statement retrieves all the account numbers and corresponding customer names whose account balance is in the range of values between `min_sal` and `max_sal`.

Slide 26

Let us understand the usage of `IS NULL` clause.



IS NULL Clause

- Used along with `WHERE` clause to check whether certain property has an associated value or not.
- Following code snippet shows the usage of `IS NULL` clause:

```

...
TypedQuery<String> query = em.createQuery(
    " SELECT a.name, a.address
    FROM Account a WHERE a.email IS NULL", String.class);

List<String> results = query.getResultList();
...

```

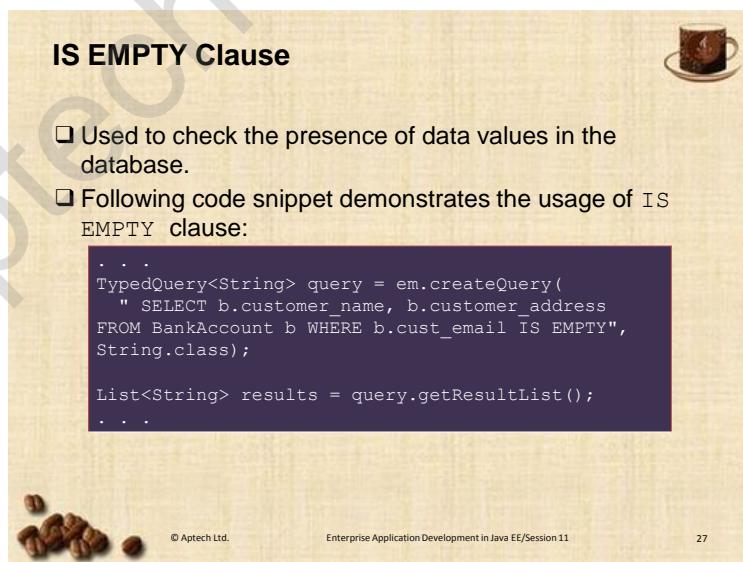
© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 26

Use slide 26 to explain the usage of `IS NULL` clause in the query.

The query in the `createQuery()` method retrieves the name and e-mail address of customers whose e-mail id value is `NULL`.

Slide 27

Let us understand the working of `IS EMPTY` clause.



IS EMPTY Clause

- Used to check the presence of data values in the database.
- Following code snippet demonstrates the usage of `IS EMPTY` clause:

```

...
TypedQuery<String> query = em.createQuery(
    " SELECT b.customer_name, b.customer_address
    FROM BankAccount b WHERE b.cust_email IS EMPTY",
    String.class);

List<String> results = query.getResultList();
...

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 27

Use slide 27 to explain `IS EMPTY` clause.

This clause is used to check whether there are any entities in the database holds the condition specified in the `WHERE` clause.

The query defined in the `createQuery()` method checks whether there are any methods whose e-mail id do not exist in the database.

Tips:

The difference between `IS NULL` and `IS EMPTY` is that the `NULL` value can be explicitly set in the database to imply that the value is not applicable to this case or it is not currently known. The `EMPTY` field signifies that the data is not present.

Slide 28

Let us understand polymorphic queries.

Polymorphic Queries

- ❑ When a JPQL query is executed on an entity class, then the query executed is applicable to all its sub classes.
- ❑ The entity in the JPQL query is placed along with the 'where' clause in the query.
- ❑ By default all queries are polymorphic.
- ❑ Following figure shows the `Account` hierarchy:

```

graph TD
    Account[Account] --> Savings[Savings account]
    Account --> Current[Current account]
  
```

- ❑ A query on `Account` entity is applicable to both the `SavingsAccount` entity and `CurrentAccount` entity.
- ❑ For example: `select customer_name from Account;`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 28

Use slide 28 to explain polymorphic queries. Polymorphic queries are queries applied on an inheritance hierarchy.

A polymorphic query on a super class is applicable on all its sub classes. According to the hierarchy given in the slide, any query on the `Account` entity is applicable to its sub classes – `Savings account` and `Current account`. This means that the instances returned by a query include instances of all the subclasses satisfying the condition.

Then, explain the example shown on the slide to understand the result of polymorphic query.

Additional References:

To get more information on Query API and its examples, refer the following link:
http://docs.oracle.com/cd/E13189_01/kodo/docs40/full/html/ejb3_overview_query.html#ejb3_overview_query_inheritance.

Slide 29

Let us understand Criteria API.

Overview of Criteria API

- ❑ Used to define dynamic queries through query-defining objects.
- ❑ Are defined by instantiating Java objects.
- ❑ Offers better integration with Java language than JPQL queries as it operates on the database in terms of objects.
- ❑ Accepts values for each clause of the query through different methods and validates them.
- ❑ Are queries based on abstract schema of persistent entities, their relationships, and embedded objects.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 29

Use slide 29 to explain Criteria API. Tell them that Criteria API is used to write dynamic queries and provides methods to make these dynamic queries compatible with object-oriented code.

The queries written using JPQL accept String parameters as query, this does not provide for checking the syntax of the SQL query to be executed on the database. This may result in runtime errors. The Criteria API removes this possibility of errors as it accepts the query as parameter of different SQL clauses. Each SQL clause is implemented as a method in Criteria API.

The queries are written based on the abstract database schema.

Slide 30

Let us understand how to create a Criteria query.

Creating Criteria Query

- ❑ Criteria API creates queries by obtaining an instance of `javax.persistence.criteria.CriteriaQuery`.
- ❑ The `createQuery()` method is used to create the query.
- ❑ Following code snippet demonstrates the creation of query:

```
. . .
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery cq = cb.createQuery();
. . .
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 30

Use slide 30 to explain how a developer can create a Criteria query.

A `CriteriaQuery` is created as an instance of `javax.persistence.criteria.CriteriaQuery`.

The `createQuery()` method can be used to create an instance of `CriteriaQuery`.

Slide 31

Let us understand query root.

Query Root

- ❑ The query root in Criteria API query is entity where the navigation initiates.
- ❑ Created by invoking `from()` method of `CriteriaQuery` interface.
- ❑ The query root can be defined as:
`Root c = cq.from(Customer.class);`
- ❑ For queries which require multiple entities at the root, the `from()` method is invoked with multiple entity classes as parameter.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 31

Use slide 31 to explain query root. Query root refers to the tables of the data on which the SQL query has to be executed. These are the tables which appear in the FROM clause of a SQL query.

The query root can be defined by passing the Entity class corresponding to the table as parameter. If the query requires multiple Entity classes, then multiple classes are added as parameters to the `from()` method.

Slides 32 and 33

Let us understand the steps involved in creating a Criteria API.

Using Criteria API 1-5

- ❑ Following are the steps involved in creating a criteria API query:

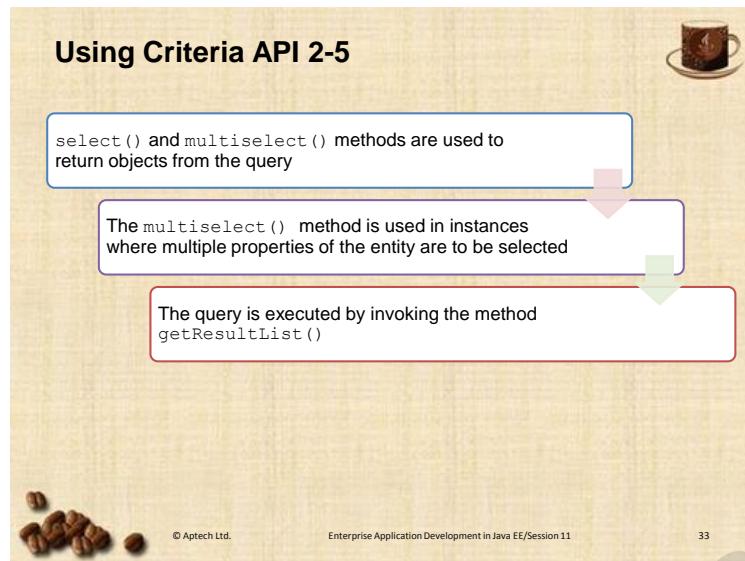
`CriteriaBuilder` object created by `EntityManager`

`CriteriaBuilder` creates `CriteriaQuery` object to create `SELECT` queries

`from()` method is used to identify the entities on which the query is defined

Each clause of the query is defined as a method, a `where()` method is used to apply conditions

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 32

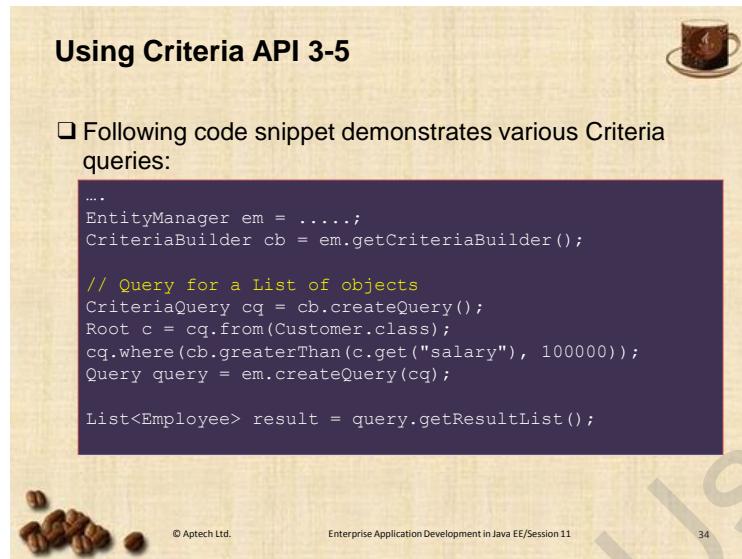


Use slides 32 and 33 to explain the steps involved in creating a Criteria API query.

1. The entities in the application are managed through the EntityManager. The EntityManager in turn creates a CriteriaBuilder object. The CriteriaBuilder object is the main interface of the Criteria API; it is responsible for creating Criteria queries.
2. The CriteriaBuilder object further creates a CriteriaQuery object. This object defines a database select query; it can enable usage of all the clauses associated with the JPQL SELECT statement. The CriteriaBuilder and CriteriaQuery classes are specific to the Criteria API. They build the context to accept queries on entities, where the entities are interpreted as objects.
3. The `from()` method of the CriteriaQuery interface is used to set the query root. The function performed by the `from()` method is similar to that of the FROM clause in a SQL query. The query root implies the entity sets from which the result set of the query has to be generated. For instance, if the query has to be executed on Account entity set, then it forms the query root in the Criteria API.
4. The clauses used along with the 'SELECT' statement are defined as methods in the Criteria API such as `where()`, instead of WHERE clause. The `where()` method is used to apply conditions on the result set.
5. The `select()` method is used to define the return object type of the query. This method is used when the query has to return only one type of value. When there are values with multiple data types to be returned by the query, then the developers have to use `multiselect()` method.
6. The `multiselect()` method is used in instances where multiple properties of the entity are to be selected.
7. The query is executed by invoking the method `getResultList()`.

Slide 34

Let us understand the queries written using the Criteria API.



Using Criteria API 3-5

Following code snippet demonstrates various Criteria queries:

```

...
EntityManager em = ....;
CriteriaBuilder cb = em.getCriteriaBuilder();

// Query for a List of objects
CriteriaQuery cq = cb.createQuery();
Root c = cq.from(Customer.class);
cq.where(cb.greaterThan(c.get("salary"), 100000));
Query query = em.createQuery(cq);

List<Employee> result = query.getResultList();

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 34

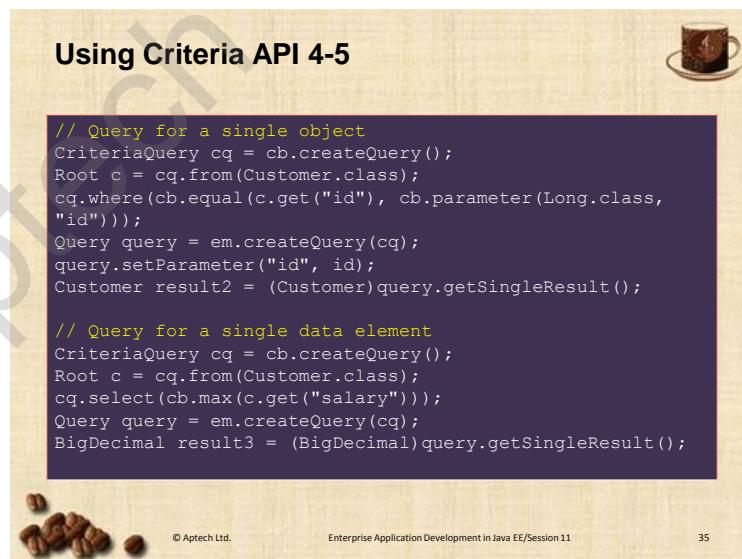
Use slide 34 to explain a query written using Criteria API.

The query root is set to Customer entity class, that is, the query is executed on `Customer` table. The condition of the WHERE is clause is defined through `WHERE` method. The condition is the salary value of the entity should be greater than 10000. This condition is defined through methods `get()` which retrieves the value and `greaterthan()` which compares the salary value with 10000.

The result of the query is retrieved through the `getResultList()` method.

Slide 35

Let us understand more about Criteria queries.



Using Criteria API 4-5

```

// Query for a single object
CriteriaQuery cq = cb.createQuery();
Root c = cq.from(Customer.class);
cq.where(cb.equal(c.get("id"), cb.parameter(Long.class,
"id")));
Query query = em.createQuery(cq);
query.setParameter("id", id);
Customer result2 = (Customer)query.getSingleResult();

// Query for a single data element
CriteriaQuery cq = cb.createQuery();
Root c = cq.from(Customer.class);
cq.select(cb.max(c.get("salary")));
Query query = em.createQuery(cq);
BigDecimal result3 = (BigDecimal)query.getSingleResult();

```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 35

Use slide 35 to explain queries used to retrieve a single object. The query root in the first query is again set to `Customer` entity class. The query expects a parameter from the user and compares this parameter against the id. If the parameter id is equal to a value in the `Customer` entity then that entity is retrieved from the application database.

The second criteria query retrieves a specific value of an entity in the application. The query root is set to the Customer entity class. The query attempts to retrieve the maximum salary value from the Customer entities through methods `get()` and `max()`.

Slide 36

Let us understand some more about Criteria queries.



Using Criteria API 5-5

```
// Query for a List of data elements
CriteriaQuery cq = cb.createQuery();
Root c = cq.from(Customer.class);
cq.select(c.get("firstName"));
Query query = em.createQuery(cq);
List<String> result4 = query.getResultList();

// Query for a List of element arrays
CriteriaQuery cq = cb.createQuery();
Root c = cq.from(Customer.class);
cq.multiselect(c.get("firstName"), c.get("lastName"));
Query query = em.createQuery(cq);
List<Object[]> result5 = query.getResultList();
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 36

Use slide 36 to explain few more Criteria queries.

The first query in the slide has the query root set to the Customer entity class. The query attempts to retrieve the first name of the Customer entity through `get()` method.

The second query in the slide retrieves data of multiple columns of the Customer entity. The query retrieves the first name and last name of the customer entity.

Slide 37

Let us understand the clauses used to manage the Criteria query results.



Managing Criteria Query Results 1-3

- ❑ The Criteria API provides methods such as `orderBy()` and `groupBy()` to implement ORDER BY and GROUP BY clauses.
- ❑ The `asc()` and `desc()` methods are used to sort the results.
- ❑ Following code snippet demonstrates the usage of `order` `By()` method:

```
...
CriteriaQuery<BankAccount> cq =
cb.createQuery(BankAccount.class);
Root<BankAccount> BA = cq.from(BankAccount.class);
cq.select(BA);
cq.orderBy(cb.desc(BA.get(BankAccount_.acc_bal)));
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 37

Use slide 37 to explain the methods provided by the Criteria API to manage the results of the query.

Clauses such as ORDER BY and GROUP BY are used to manage the results of an SQL query. Criteria API provide methods to implement the function of ORDER BY and GROUP BY clauses.

Criteria API provides orderBy(), groupBy(), asc(), and desc() methods to manage the results of the query.

Explain the usage of orderby() method in the Criteria query.

Slide 38

Let us understand the usage of groupBy() method.

Managing Criteria Query Results 2-3

Following code snippet demonstrates the usage of groupBy() method:

```
....  
CriteriaQuery<BankAccount> cq =  
cb.createQuery(BankAccount.class);  
Root<BankAccount> BA = cq.from(BankAccount.class);  
cq.select(BA);  
cq.orderBy(cb.desc(BA.get(BankAccount_.acc_bal)));  
....
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 38

Use slide 38 to demonstrate the usage of groupBy() method.

Like orderBy(), groupBy() is also implemented as a method. In the code snippet, the Customer entities are grouped according to the account balance.

The grouped data is arranged in descending order.

Slide 39

Let us understand the usage of having () method.

Managing Criteria Query Results 3-3

Following code snippet demonstrates the usage of `having()` method:

```
...
CriteriaQuery<BankAccount> cq =
cb.createQuery(BankAccount.class);
Root<BankAccount> BA = cq.from(BankAccount.class);
cq.groupBy(BA.get(BankAccount_.acc_type));
cq.having(cb.in(BA.get(BankAccount_.acc_type)).value(savings));
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 39

Use slide 39, explain the usage of `having()` method. HAVING clause in SQL queries is used along with GROUP BY clause. It is used to apply a condition on groups of values in the database. Therefore, `having()` method in Criteria API is used along with `groupBy()` method.

In the given query, the data is grouped on the basis of types of accounts. The query retrieves all the entities which hold a Savings account.

Slide 40

Let us understand the steps used in executing a Criteria query.

Executing Criteria Query

CriteriaBuilder prepares for the execution of the query

A TypedQuery object must be created which is the return type of the query

TypedQuery object must be defined through the `createQuery()` method of EntityManager

`getSingleResult()` and `getResultList()` methods are used for executing queries

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 40

Use slide 40 to explain the steps for executing Criteria query.

Before executing the query, the `CriteriaBuilder` should prepare for the execution of the query. A `TypedQuery` object must be created with the type of query result. The `TypedQuery` object must be defined through the `createQuery` method invoked by the `EntityManager`.

Following statement creates an instance of the TypedQuery.

```
TypedQuery<BankAccount> query = em.createQuery(cq);
```

The methods used for executing queries are `getSingleResult()` and `getResultList()`. The `getSingleResult()` method returns only one result as output of the query. The `getResultList()` method returns a collection of entities as output of the query.

Slide 41

Let us understand how to join queries written using Criteria API.

Querying Relationships Using Joins

- Criteria queries implement join operation on entities through `join()` method.
 - The `join` operation implies comparing the values of one attribute of an entity with a similar attribute of another entity.
- Following code snippet demonstrates the usage of `join()` method:

```
....  
CriteriaQuery<BankAccount> cq =  
cb.createQuery(BankAccount.class);  
Root<BankAccount> BA = cq.from(BankAccount.class);  
Join<BankAccount, Customer> result =  
cq.join(BankAccount_.Customer);  
....
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 41

Use slide 41 to explain how developers can write joined queries using Criteria API. The query root can be set to only one entity class. The second entity class with which the JOIN operation has to be performed is provided as parameter to the `join()` method.

Slide 42

Let us understand the Metamodel API.

Using Metamodel API 1-4

- MetaModel API works along with the Criteria API to model persistence entity classes.
- JPA Metamodel API enables the developers to understand the database schema of the application data.
- Metamodel comprises information about the mapped attributes for an entity class, their corresponding data types, and so on.
- The Metamodel classes corresponding to the entity classes are of the form:
 - `javax.persistence.metamodel.EntityType<T>`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 42

Use slide 42 to explain Metamodel API. It enables developers to understand the database schema of the PersistenceContext.

It enables retrieval of details on managed classes and persistent fields and their relationship in Criteria queries. The Metamodel comprises information about the mapped attributes for an entity class, their corresponding data types, and so on.

For each entity class in the application a corresponding Metamodel class can be created.

Slides 43 and 44

Let us understand the metamodel class for an entity class.

Using Metamodel API 2-4



- ❑ The developer can use Metamodel API to create the metamodel of the entities managed in the persistence unit of the application.
- ❑ Following code snippet shows an entity class:

```
...
@Entity
public class BankAccount {
@Id
protected Long acc_no;
protected String account_holder_name;
protected String acc_type;
protected Long acc_bal
...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 43

Using Metamodel API 3-4



- ❑ Following code snippet shows the metamodel class for the BankAccount entity class:

```
...
@Static Metamodel(BankAccount.class)
public class BankAccount_ {
public static volatile SingularAttribute<BankAccount,
Long> acc_no;
public static volatile SingularAttribute<BankAccount,
String> account_holder_name;
public static volatile SingularAttribute<BankAccount,
String> acc_type;
public static volatile SingularAttribute<BankAccount,
Long> acc_bal;
...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 44

Use slides 43 and 44 to explain the entity class. This Metamodel class has the same class name as entity class followed by an underscore. In addition, the Metamodel class has attributes which correspond to the properties of the entity class.

Slide 43 shows an entity class and slide 44 shows the corresponding metamodel class.

Slide 45

Let us understand the method which allows creation of metamodel class.

Using Metamodel API 4-4

- A Metamodel class can be generated through `getModel()` method.
- Following code snippet demonstrates the usage of `getModel()` method to obtain the `BankAccount` entity:

```
CriteriaQuery cq =
    cb.createQuery(BankAccount.class);
Root<BankAccount> c = cq.from(BankAccount.class);
EntityType<BankAccount> account_ = c.getModel();
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 45

Use slide 45 to explain the creation of metamodel class from the entity class using the `getModel()` method.

In-Class Question:

After you finish explaining the creation of metamodel class, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



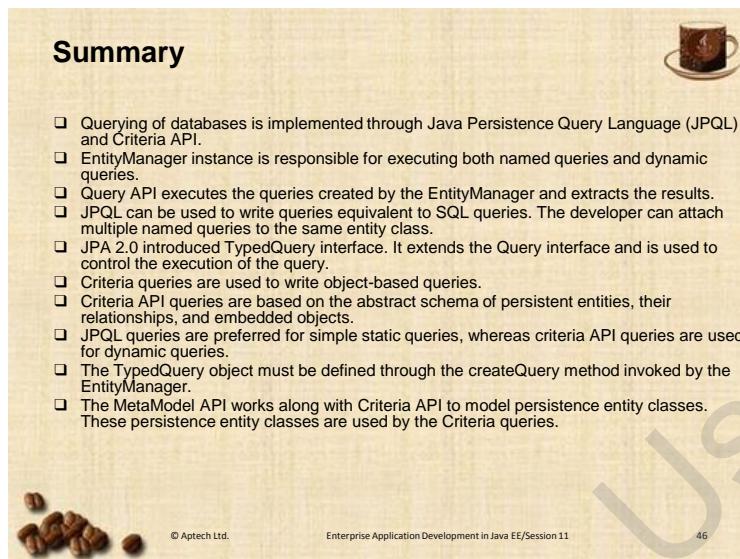
Which technique of writing queries is prone to syntax errors?

Answer:

JPQL queries, because the method interprets the query as a string and the syntax of the string is only checked during the execution phase.

Slide 46

Let us summarize the session.



Summary

- ❑ Querying of databases is implemented through Java Persistence Query Language (JPQL) and Criteria API.
- ❑ EntityManager instance is responsible for executing both named queries and dynamic queries.
- ❑ Query API executes the queries created by the EntityManager and extracts the results.
- ❑ JPQL can be used to write queries equivalent to SQL queries. The developer can attach multiple named queries to the same entity class.
- ❑ JPA 2.0 introduced TypedQuery interface. It extends the Query interface and is used to control the execution of the query.
- ❑ Criteria queries are used to write object-based queries.
- ❑ Criteria API queries are based on the abstract schema of persistent entities, their relationships, and embedded objects.
- ❑ JPQL queries are preferred for simple static queries, whereas criteria API queries are used for dynamic queries.
- ❑ The TypedQuery object must be defined through the createQuery method invoked by the EntityManager.
- ❑ The MetaModel API works along with Criteria API to model persistence entity classes. These persistence entity classes are used by the Criteria queries.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 11 46

Using slide 46, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

11.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Concurrency, Listeners, and Caching that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 12 – Concurrency, Listeners and Caching

12.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

12.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe concurrency utilities provided by Java EE
- Explain locking techniques used to enable concurrency in entities
- Describe different locking modes
- Explain entity lifecycle callback events
- Explain how to inject external listeners to handle lifecycle callback events
- Explain caching techniques
- Explain how to specify caching modes in applications

12.1.2 Teaching Skills

To teach this session successfully, you should be aware of the concurrency control techniques used by enterprise applications. You should know why these techniques are required in enterprise applications, what kind of overhead do they place on application performance. You should be able to highlight the purpose of concurrency control techniques on databases.

The session discusses various locking mechanisms and caching methods used by enterprise applications.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

The slide has a parchment-like background with a coffee cup icon in the top right corner and coffee beans at the bottom left. The title 'Objectives' is in bold. A list of nine items follows, each starting with a square checkbox.

Objectives

- Describe concurrency utilities provided by Java EE
- Explain locking techniques used to enable concurrency in entities
- Describe different locking modes
- Explain entity lifecycle callback events
- Explain how to inject external listeners to handle lifecycle callback events
- Explain caching techniques
- Explain how to specify caching modes in applications

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 2

Use slide 2 and tell the students that they will be introduced to concurrency control mechanisms in the application. Tell the students that they will also learn the different locking mechanisms provided by the Java EE platform which can be used for concurrency control also. Tell them that they will learn about the APIs provided by Java EE to implement locking mechanisms in the application.

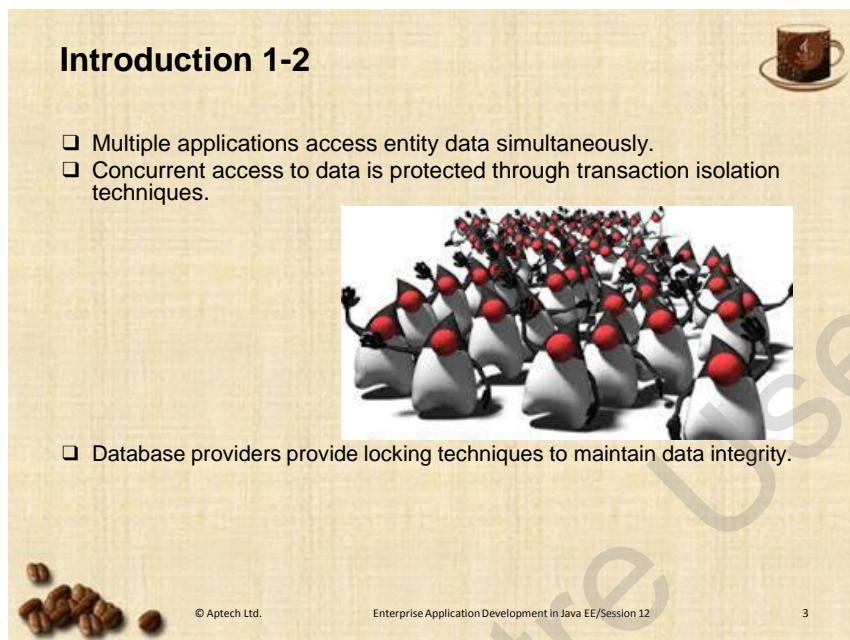
Tell them that they will also learn about caching techniques used by applications and about cache which is high speed memory which stores frequently used data. Placing frequently used data in high speed memory will improve the application performance. Various caching techniques and caching modes used in the application will also be explained.

12.2 In-Class Explanations

Slide 3

Let us understand what is meant by concurrency in applications.

Introduction 1-2



- ❑ Multiple applications access entity data simultaneously.
- ❑ Concurrent access to data is protected through transaction isolation techniques.



❑ Database providers provide locking techniques to maintain data integrity.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 3

Use slide 3 to explain the meaning of concurrency in enterprise applications. Enterprise applications are multi-user systems; multiple clients can simultaneously access a single application. When multiple clients are accessing the data there is a risk of data integrity violation. Concurrent access to the database is always protected with the transaction isolation techniques.

Transaction isolation techniques include locking mechanisms and corresponding protocols. Java Persistence API which is part of Java EE platform provides various implementations of acquiring locks on data items in the application. The modern databases also offers the in-built concurrency features.

Slide 4

Let us understand the locking mechanisms provided by the JPA.

Introduction 2-2

- ❑ JPA specification has defined two important mechanisms to tune concurrent access to entities:
 - Optimistic locking
 - Explicit read and write locks

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 4

Use slide 4 to explain the mechanisms provided by JPA to implement concurrent access techniques.

JPA provides optimistic locking mechanism and a mechanism to acquire read and write locks explicitly. The mechanism in which read and write locks are explicitly acquired is also known as pessimistic locking.

In case of optimistic locking the application does not acquire any lock initially, it executes the operation and before committing the operation it checks for data consistency and if necessary, acquires the lock and performs the operations.

In case of pessimistic locking the application acquires locks before the execution of operation starts. So, even if the lock is not required, still the application will acquire the lock and then execute the operations.

Slides 5 and 6

Let us understand optimistic locking.

Optimistic Locking 1-2



- Does not extensively acquire locks.
- Provides high degree of concurrent access.
- Checks the consistency of the data read from the database before committing the transaction.
- JPA specification configures transaction isolation level to READ COMMITTED by default.
- javax.persistence.Version attribute of entities used to check the consistency of the data.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 5

Optimistic Locking 2-2



Working of optimistic locking:

- Before the transaction commits, optimistic locking technique ensures the following:
 - If an update operation is performed on an old version of the entity instance corresponding to the database table row whose data is already being updated by an entity instance running in other transaction boundary.
 - Then, an exception is thrown. The exception named OptimisticLockException is thrown by the persistence provider when such a conflict occurs.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 6

Use slides 5 and 6 to explain optimistic locking of data by applications.

An application can acquire a lock when it has to read or write data. When the application has to perform a read operation, it acquires a shared lock where multiple transactions can read an entity. If the application has to perform a write operation then the application would acquire an exclusive lock. In case of exclusive lock no other transactions can access the data that means the other application can neither read nor write the locked data.

When a data item is locked by a transaction, then the transactions which have to perform a write operation on the data have to wait for the shared lock to be released. Similarly, when a data entity is

exclusively locked by a transaction then other operations/transactions trying to read/write the data have to wait until the exclusive lock is released.

The application performance may decrease or it may become slow, when operations or transactions wait to acquire locks.

Optimistic locking technique does not acquire locks for a transaction, instead it carries out the transaction and checks the version of the data accessed before committing the transaction. If the data has changed since the transaction has accessed it and it is inconsistent with the current state, then the transaction is rolled back otherwise it is committed.

It provides high level of transaction concurrency as it does not acquire any locks and hence multiple transactions can simultaneously read or write data.

JPA defines various transaction isolation levels, the default transaction isolation level is READ COMMITTED. According to this transaction isolation level, the application can read only committed data. Every data item is associated with a **javax.persistence.Version** attribute which is used to identify whether the data has changed since it was last accessed.

The Version attribute is updated when the data time it is associated with has changed. The transaction before committing checks whether all the data it has accessed has changed by accessing the Version attribute.

The exception named **OptimisticLockException** is thrown by the persistence provider when a Version conflicts occurs.

Slide 7

Let us understand the Version attribute.

Version Attribute 1-2

Persistence provider modifies the `Version` attribute whenever the entity state data is modified.

Following are the requirements of `Version` attribute:

- Each entity class is associated with only one `Version` attribute
- Only persistence provider can set or update the value of `Version` attribute
- `Version` attribute should be present in the primary table of the database
- `Version` attribute can be of type - `int`, `Integer`, `long`, `Long`, `short`, `Short`, or `java.sql.Timestamp`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 7

Use slide 7 to explain Version attribute provided by the JPA API. Transactions use the Version attribute to understand whether the entity has changed since it was last accessed by the transaction.

Every entity is associated with a Version attribute; the value of the Version attribute is incremented when a transaction changes it. The value of the Version attribute cannot be set by the bean classes and user methods, only persistence provider can modify the Version attribute.

The Version attribute in the entity tables can be of type – `int`, `Integer`, `long`, `Long`, `short`, `Short`, or `java.sql.Timestamp`.

Slide 8

Let us understand how Version attribute is included in the Entity class.

Version Attribute 2-2




Following code snippet shows the Version attribute provided to an entity class:

```
public class Employee {
    @ID
    int id;

    @Version
    int version;
    ...
}
```

The inclusion of version attribute marked with the annotation @Version instructs the persistence provider to check the entity instance for:

- Any concurrent modifications and increments of the version attribute in each update operation.

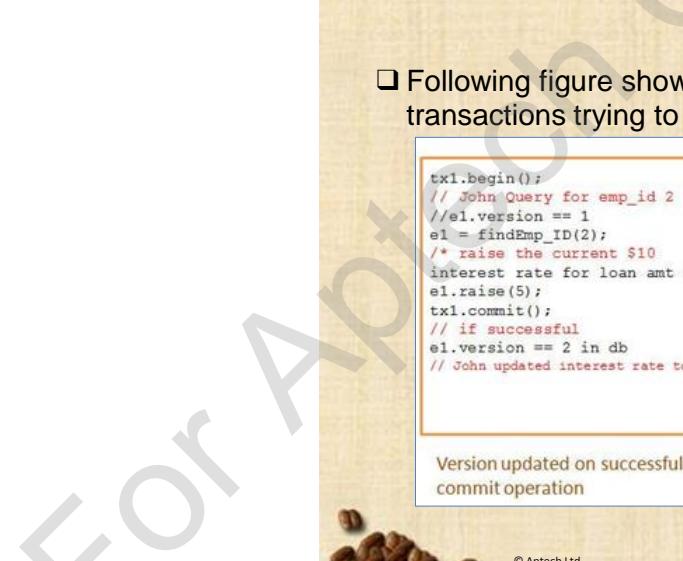
© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 8

Use slide 8 to explain how Version attribute can be included in the Entity class. @Version attribute is used to define Version attribute in the class. Version attribute is updated whenever there is an update to the entity.

Slide 9

Let us demonstrate an example of concurrent transactions.

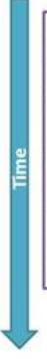
Concurrent Transactions

Following figure shows the example of two concurrent transactions trying to update the Employee entity:

```
tx1.begin();
// John Query for emp_id 2
// e1.version == 1
e1 = findEmp_ID(2);
/* raise the current $10
interest rate for loan amt */
e1.raise(5);
tx1.commit();
// if successful
e1.version == 2 in db
// John updated interest rate to 15
```

Version updated on successful commit operation



```
tx2.begin();
// Jill Query for emp_id 2
// e1.version == 1
e1 = findEmp_ID(2);
/* raise the current $10
interest rate for loan amt */
e1.raise(2);

// if e1 == 1 version in db
tx2.commit();

// if unsuccessful
throw OptimisticLockException();
```

Exception raised as Version already updated

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 9

Use slide 9 to demonstrate an example of concurrent transactions.

Slide 9 shows two transactions tx1 and tx2. Both the transactions are retrieving information of employee with employee ID 2.

Both the transactions invoke `raise()` method on the entity e1.

The transaction tx1 performs a commit and modifies the Version attribute of the entity. Transaction tx2 has read the data when the Version attribute value is 1, hence it can commit the transaction only if the version value is 1. When it discovers that the version value has changed then it rolls back the transaction throwing an exception.

Slide 10

Let us understand pessimistic locking.

Pessimistic Locking 1-3

- ❑ Pessimistic locking acquires locks extensively on entity data.
- ❑ Pessimistic locking techniques ensure that:
 - No other transaction can read or update the entity instance until the current transaction is committed.
 - If the pessimistic lock is applied as an exclusive lock, then only the holding transaction can update or delete.
- ❑ A pessimistic lock mode can be set to:
 - PESSIMISTIC_READ
 - PESSIMISTIC_WRITE
 - PESSIMISTIC_FORCE_INCREMENT

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 10

Use slide 10 to introduce pessimistic locking. Pessimistic locking mechanism acquires locks on all the entities of the transaction before beginning the transaction. This mechanism is used when the transaction requires high reliability. Pessimistic locking reduces the throughput of the transactions executed by an application.

A pessimistic lock can be acquired on entity data by setting the lock mode of the entity instance to PESSIMISTIC_READ, PESSIMISTIC_WRITE, or PESSIMISTIC_FORCE_INCREMENT.

Slide 11

Let us understand various methods and properties used for pessimistic locking.

Pessimistic Locking 2-3

- ❑ An application cannot wait indefinitely to acquire a pessimistic lock.
- ❑ A timeout period is set through `javax.persistence.lock.timeout` property.
- ❑ The lock timeout property can be set by the EntityManager.
- ❑ `Query.setLockMode` and `TypedQuery.setLockMode` methods are used to set the timeout.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 11

Use slide 11 to explain various methods which are used for pessimistic locking. As a transaction cannot indefinitely wait to acquire locks, a timeout period is associated with the locking mechanism. The transaction which is trying to acquire the lock will wait for time period defined by the timeout property.

Developers can set the value of the timeout period through EntityManager class.

`setLockMode()` method is used to acquire locks on entities.

Slide 12

Let us understand different ways of setting the timeout period for acquiring locks.

Pessimistic Locking 3-3

Following is the order followed in determining the timeout period:

- Timeout period set through instances of EntityManager and Query
- Timeout period set through @NamedQuery annotation
- Timeout period set through createEntityManagerFactory()
- Timeout period defined in the deployment descriptor

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 12

Use slide 12 to explain different methods of setting the timeout period for the application.

The lock timeout property can be set through the `EntityManager.Query.setLockMode` and `TypedQuery.setLockMode` methods that can be used to set the lock timeout period. The timeout period can also be set through the deployment descriptor and as an attribute to the `@NamedQuery` annotation.

When the timeout period is set at multiple places, then the timeout period is determined as per the following order:

1. If the timeout period is set using the instances of `EntityManager` or `Query` methods then, this value is considered above all other methods.
2. If the timeout is not set by `EntityManager` or `Query` instances and it is specified through `@NamedQuery` annotation then this value is considered by the persistence provider.
3. If none of the earlier methods are used and the timeout period is set through the `createEntityManagerFactory()` method then this value of timeout is considered.
4. If the timeout period is not defined anywhere in the application code and defined in the deployment descriptor then this value is the final timeout value.

Slides 13 and 14

Let us understand different lock modes offered by JPA.

Lock Modes 1-6




JPA provides lock modes which are layered on the **Version** attribute.

Following table shows various lock modes provided by JPA:

Locking Mode	Description
OPTIMISTIC	This mode acquires optimistic read locks on all the entities used by the application with corresponding version attributes.
OPTIMISTIC_FORCE_INCREMENT	This mode acquires optimistic read locks on all the entities required by the application and version attributes. It force increments on the version attribute value.
PESSIMISTIC_READ	This mode acquires a long term read lock on the data accessed by the application. This lock is acquired to prevent data to be accessed or modified by other applications. Other applications can read this data when the pessimistic read lock is held on the entity data. The pessimistic read lock can be upgraded to a pessimistic write lock.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 13

Lock Modes 2-6




Locking Mode	Description
PESSIMISTIC_WRITE	Pessimistic write lock is a long term write lock acquired on entity data. This lock prevents other applications from both reading and writing the locked entity data.
PESSIMISTIC_FORCE_INCREMENT	This mode of lock obtains a long term lock on the entity data and increments the value of the version attribute. This lock prevents the data from being modified by other applications.
READ	This mode represents an acquired optimistic read lock.
WRITE	This mode represents an optimistic write lock acquired on the entity data. It forces to increment the version attribute.
NONE	No locking mode is used on the database.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 14

Use slides 13 and 14 to explain different lock modes that can be defined through JPA. The READ lock is a shared lock and WRITE lock is an exclusive lock.

Provide appropriate examples to the students to better explain the application of different lock modes.

Slide 15

Let us understand different methods of acquiring locks.

Lock Modes 3-6

Locking can be specified at multiple levels using the following methods:

- `find()`, `refresh()` and `lock()` methods of Entity Manager
- `setLockMode()` method of Query
- `lockMode` attribute of `@NamedQuery` annotation

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 15

Use slide 15 to introduce different classes provided by the Java EE platform through which applications can acquire locks.

`EntityManager` class is used to manage all the entities present in the application. This class provides `find()`, `refresh()`, and `lock()` methods to manage locks in an application.

`setLockMode()` of the `Query` interface is used to set locks on the data entity.

`@NamedQuery` annotation has an attribute lock mode which defines the type of lock to be acquired before executing the query.

Tips:

The `find()` method is used to retrieve an entity based on its attributes such as primary key. The `refresh()` method refreshes the changes made to the entity of the database. The `lock()` method acquires the lock on the entity.

Slides 16 and 17

Let us understand the syntax for using the methods.

Lock Modes 4-6

- Following code snippet shows the usage of **lock()** method:

```
...
EntityManager em = ...;
Customer c = ...;
em.lock(c, LockModeType.OPTIMISTIC);
...
```

- Following code snippet shows the usage of **find()** method:

```
EntityManager em = ...;
String customerPK = ...;
Customer C = em.find(Customer.class, customerPK,
LockModeType.PESSIMISTIC_WRITE);
...
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

16

Lock Modes 5-6

- Following code snippet shows the usage of **refresh()** method to modify the currently existing lock:

```
EntityManager em = ...;
String customerPK = ...;
Customer C = em.find(Customer.class, customerPK);
...
em.refresh(C,
LockModeType.OPTIMISTIC_FORCE_INCREMENT);
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

17

Use slides 16 and 17 to demonstrate the syntax of different methods provided by the EntityManager class.

Slide 16 shows the usage of **lock()** and **find()** methods.

Slide 17 shows the usage of **refresh()** method.

Tips:

Following are the overloaded prototypes of find() method:

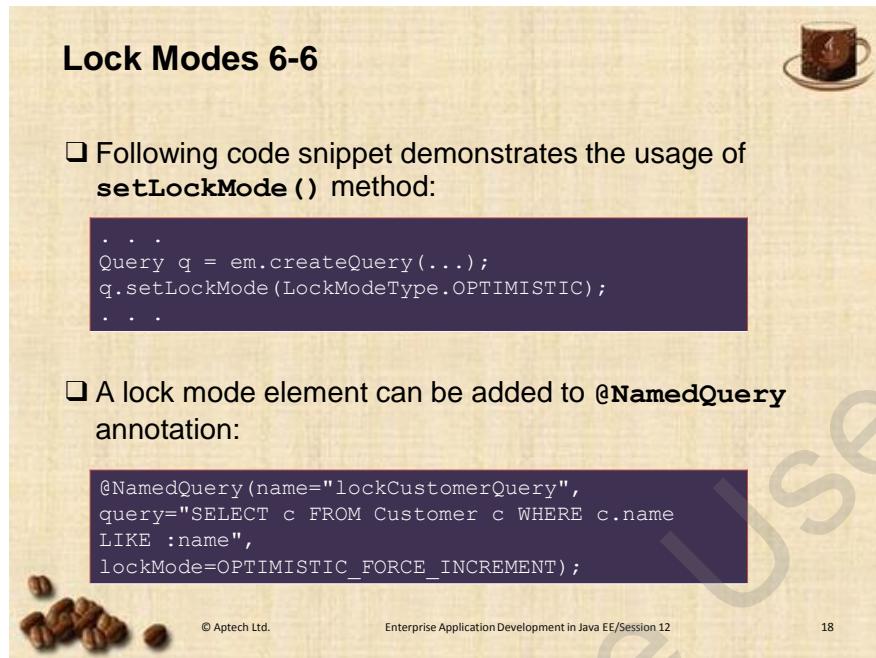
```
find(Class<T> entityClass, Object primaryKey)  
find(Class<T> entityClass, Object primaryKey, LockModeType lockMode)  
find(Class<T> entityClass, Object primaryKey, LockModeType lockMode,  
Map<String, Object> properties)  
find(Class<T> entityClass, Object primaryKey, Map<String, Object>  
properties)
```

Following are the overloaded prototypes of lock() method:

```
lock(Object entity, LockModeType lockMode)  
lock(Object entity, LockModeType lockMode, Map<String, Object>  
properties)
```

Slide 18

Let us understand how to use lock modes to set locks to entities.



Lock Modes 6-6

- ❑ Following code snippet demonstrates the usage of `setLockMode()` method:

```
...  
Query q = em.createQuery(...);  
q.setLockMode(LockModeType.OPTIMISTIC);  
...
```
- ❑ A lock mode element can be added to `@NamedQuery` annotation:

```
@NamedQuery(name="lockCustomerQuery",  
query="SELECT c FROM Customer c WHERE c.name  
LIKE :name",  
lockMode=OPTIMISTIC_FORCE_INCREMENT);
```

Use slide 18 to demonstrate the code required to use the `setLockMode()` method of Query interface to set locks on data entities.

Developers can also set a lock on data entity through `@NamedQuery` annotation. Slide 18 clearly shows how to use the `NamedQuery`.

Slide 19

Let us understand Callbacks and Listeners.

Entity Callbacks and Listeners

❑ An entity contains a predefined set of lifecycle events that are triggered on the execution of methods called from EntityManager or Query API.

JPA allows developers to setup callback methods on entity classes.

- They are invoked when the entity instances are notified about the events.

Developers can also register separate listener classes on the callback events.

- Entity listener class methods are invoked in response to life cycle events on an entity.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 19

Use slide 19 to explain entity callbacks and listeners. In an enterprise application an entity can be in a locked state where it is being accessed by some transaction in the application or in persisted state where the changes made to the entity by a transaction are committed to the application hard disk.

JPA allow developers to set up callbacks and listeners on entities when they enter either a locked or persisted state. These callback methods or listeners may further implement operations such as logging, and so on. For example, the `persist()` method triggers database events. While doing so, it may be required to log the interactions performed on the database rows.

Callbacks set on an entity are notified whenever there is an event with respect to the entity. Listeners respond to various events occurring on the entity.

Slide 20

Let us understand various callbacks on entities.

Entity Callback Events

- ❑ Callback methods are defined within the entity class.
- ❑ Following code snippet shows the implementation of callback methods in an entity class:

```
@Entity
public static class MyEntity {
    @PrePersist void onPrePersist() {}
    @PostPersist void onPostPersist() {}
    @PostLoad void onPostLoad() {}
    @PreUpdate void onPreUpdate() {}
    @PostUpdate void onPostUpdate() {}
    @PreRemove void onPreRemove() {}
    @PostRemove void onPostRemove() {}
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 20

Use slide 20 to explain different callback methods provided by JPA on entities.

- `PrePersist()` method is used to define any required operations before the entity is persisted onto permanent storage.
- `PostPersist()` method is used to define any required operations after the entity is persisted onto permanent storage.
- `onPostLoad()` method is used to define operations after the entity is loaded into the application container.
- `onPreUpdate()` method has operations to be executed before the entity is updated.
- `onPostUpdate()` method has operations to be executed after the entity is updated.
- `onPreRemove()` method has operations to be executed before removing the entity.
- `onPostRemove()` method has operations to be executed after removing the entity.

Slides 21 and 22

Let us understand the usage of lifecycle callback methods of entities with the help of an example.

Lifecycle Callback Methods 1-2



Following code snippet shows the usage of lifecycle callback methods:

```
@Entity
@EntityListeners(com.acme.AlertMonitor.class)
public class Account {
    Long accountId;
    Integer balance;
    boolean preferred;
    @Id
    public Long getAccountId() { ... }
    ...
    public Integer getBalance() { ... }
    ...
    public void deposit(Integer amount) { ... }
    public Integer withdraw(Integer amount) throws NSFException
    { ... }
    ...
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 21

Lifecycle Callback Methods 2-2



```
@PrePersist
protected void validateCreate() {
    if (getBalance() < MIN_REQUIRED_BALANCE)
        throw new AccountException("Insufficient balance to open an
account");
}
@PostLoad
protected void adjustPreferredStatus() {
    preferred = (getBalance() >=
    AccountManager.getPreferredStatusLevel());
}
public class AlertMonitor {
    @PostPersist
    public void newAccountAlert(Account acct) {
        Alerts.sendMarketingInfo(acct.getAccountId(), acct.getBalance());
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 22

Use slides 21 and 22 to demonstrate an example which uses lifecycle callback methods. Slide 21 shows an entity class Account with various properties and methods.

Slide 22 shows the lifecycle callback methods defined by the developer. `validateCreate()` is a `PrePersist` method. According to the application context, the account should have a minimum required balance to create an entity of account.

The `PostLoad` method has all the operations required to be executed after the entity is loaded into the container. `AlertMonitor()` is a `PostPersist` method where the application can send marketing alerts to the given entity.

Slides 23 and 24

Let us understand entity listeners.

Entity Listeners 1-3



Entity listeners are classes that are intercept entity callback events.

They are external classes that are attached to the entity class through annotations.

Following code snippet shows the listener class annotated with the lifecycle callback methods:

```
public class Auditor {
    @PostPersist
    void postInsert(final Object entity)
    {
        System.out.println("Inserted entity: " +
            entity.getClass().getName());
    }
    . . .
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 23

Entity Listeners 2-3



```
@PostLoad
void postLoad(final Object entity)
{
    System.out.println("Loaded entity: " +
        entity.getClass().getName());
}
. . .
}
```

Declares the class `Auditor` that defines two methods namely, `PostInsert()` and `PostLoad()`.

Both the methods are annotated with the lifecycle callback event annotations.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 24

Use slides 23 and 24 to explain entity listeners. Entity listeners are classes defined in the application which can intercept callback events such as `PrePersist`, `PostLoad`, and so on. Listeners have the operations to be executed when these callback events occur.

Listeners are defined as classes which are connected to the entity classes through annotations.

Slide 23 shows a listener method to be executed when a `PostPersist` event occurs in the application. Similarly, slide 24 shows a listener method for a `PostLoad` event.

Slide 25

Let us demonstrate how entity classes are linked with callback listener classes.

Entity Listeners 3-3

- ❑ Following code snippet shows the listener class that can be applied using `@javax.persistence.EntityListeners`:

```
@Entity
@EntityListeners ({Auditor.class})
public class EntityListenerEmployee
{
    ...
}
```

- ❑ By using the `@EntityListeners` annotation on the `EntityListenerEmployee` entity class, any callback methods within those entity listener classes will be invoked,

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 25

Use slide 25 to explain how entity listeners are connected with corresponding entity classes.

Slide 25 shows an entity class `EntityListenerEmployee` on which an entity listener `Auditor.class` is defined. EntityListeners for a class can also be mapped through deployment descriptors. The users can invoke any entity listener.

Slide 26

Let us understand default entity listener.

Default Entity Listeners



❑ Are a set of default entity listeners that are applied to every entity class in the persistence unit by using `<entity-listeners>` and `<entity-mappings>`.

❑ Following code snippet shows the default entity listener:

```
<entity-mappings>
<entity-listeners>
    <entity-listener class="com.listener.Auditor">
        <post-persist name="postInsert"/>
        <post-load name="postLoad"/>
    </entity-listener>
</entity-listeners>
</entity-mappings>
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 26

Use slide 26 to explain how entity listeners can be mapped onto entities using a deployment descriptor. The XML code shown on slide 26 clearly shows how to build the house. The deployment descriptor has `<entity-listeners>` tag, which further defines the entity-listener class. The entity-listener class has methods to listen post-persist and post load events which are defined through appropriate tags.

Slide 27

Let us understand the order of callback methods for entities.

Invocation Order of Callback Methods

Following are the rules which define the invocation order of callback methods:

- Callback methods on listener classes are invoked before the callback methods of entity classes.
- Default listeners are handled before the listeners of the top level entity class listeners until listeners of the actual entity class.
- Internal callback methods are invoked starting at the top level entity class down the hierarchy until the callback methods in the actual entity class are invoked.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 27

Use slide 27 to explain the order of invocation of callback methods in the application.

If more than one callback method has to be invoked for a lifecycle event, then the invocation order is based on the following rules:

- All the external callback methods (which are defined in listeners) are invoked, before the internal callback methods (which are defined in entity classes).
- Default listeners are handled first. When the application has a hierarchy of classes the listeners of the top level entity class, and then down the hierarchy until listeners of the actual entity class. If there is more than one default listener or more than one listener at the same level in the hierarchy, the invocation order follows the definition order.
- Internal callback methods are invoked starting at the top level entity class and then down the hierarchy until the callback methods in the actual entity class are invoked.

Slide 28

Let us understand caching in enterprise applications.

Using Second Level Cache in Persistence Applications 1-4

- ❑ Caching is a technique used to improve application performance.
- ❑ Caching optimizes the execution of expensive database calls.
- ❑ Caching can be implemented in applications at two levels:
 - First level cache – also known as `EntityManager` cache
 - Second level cache – can span across multiple `EntityManagers`.
- ❑ Applications use `javax.persistence.Cache` interface to manage the second level cache.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 28

Use slide 28 to explain second level caching in enterprise applications. Applications use caching technique to improve the application performance. First level cache is known as EntityManager cache, this cache is maintained by the EntityManager for a specific database. Second level cache is used where the cache can span across multiple entity managers. Java EE provides `javax.persistence.Cache` to implement this function. Explain to them various classes provided by Java as part of the namespace `javax.persistence`.

Slide 29

Let us understand various caching options provided by JPA.

Using Second Level Cache in Persistence Applications 2-4



Following table shows the cache modes defined in JPA:

Cache Mode Setting	Description
ALL	All the data accessed from the persistence provider is stored in the second level cache.
NONE	None of the data accessed from the persistence provider is stored in the second level cache.
ENABLE_SELECTIVE	This enables caching of the entities which are explicitly annotated with <code>@Cacheable</code> annotation.
DISABLE_SELECTIVE	This enables caching of all the entities except those entities which are explicitly annotated with <code>@Cacheable (false)</code> annotation.
UNSPECIFIED	When the cache mode is not specified then the caching behavior is set to the default behavior of the persistence provider.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 29

Use slide 29 to explain different caching modes provided by JPA. JPA provides five modes of caching as shown on the slide. The cache mode has to be appropriately selected based on other factors of the application to improve the application performance efficiency. These modes are set either through the annotation `@Cacheable` or through the deployment descriptor.

Slide 30

Let us demonstrate the usage of @Cacheable annotation.

Using Second Level Cache in Persistence Applications 4-4

- ❑ Applications can define whether the entity data can be cached or not through the javax.persistence.Cacheable annotation.
- ❑ Following code snippet demonstrates the usage of Cacheable annotation:

```
@Cacheable(true)
@Entity
public class Customer{ ... }

.....
@Cacheable(false)
@Entity
public class Loan_Application{ ... }
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 30

Use slide 30 to explain the usage of @Cacheable annotation. According to the code shown, the entity Customer is cached and the entity Loan_Application is not cached. Explain to the students the advantages and disadvantages of using caching features in Java and also how it affects the performance of the current application.

Slide 31

Let us understand how to set the cache mode through deployment descriptor.

Specifying the Cache Mode Settings



- ❑ Cache mode settings are done through the deployment descriptor.
- ❑ Following code snippet shows how to define the cache mode in the deployment descriptor:

```
...  
<persistence-unit name="BankPU" transaction-type="JTA">  
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>  
    <jta-data-source>java:comp/DefaultDataSource</jta-data-source>  
    <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>  
</persistence-unit>  
...
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 31

Use slide 31 to explain how the cache mode is set through the deployment descriptor. The `jta-data-source` tag defines the database for which the cache mode of the application has to be defined. `shared-cache-mode` tag defines the caching mode to be used.

Slide 32

Let us understand how to specify the cache retrieve mode and store mode.

Specifying the Cache Retrieval and Store Mode

1-3

- ❑ Developer has to set `javax.persistence.cache.retrieveMode` and `javax.persistence.cache.storeMode` through the `EntityManager`.
- ❑ Cache retrieval mode is an enumerated type with values **USE** and **BYPASS**.
- ❑ Cache store mode defines how data is stored on the database.
- ❑ Cache store mode can assume one of the three values – **USE**, **BYPASS**, and **REFRESH**.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 32

Use slide 32 to explain how to set cache retrieve mode and store mode that can be set to an application.

Data can be coached at two instances in an application, one when the data is retrieved from permanent storage into the container and when the data is stored onto the permanent storage from the application container.

Cache retrieval mode can be set to either **USE** or **BYPASS**. This cache retrieval mode is an enumerated type. When the mode is set to **USE** it implies that the data should be retrieved from the database by storing a copy in the cache. When the retrieval mode is set to **BYPASS** then a copy of the data retrieved from the database is not written to the cache.

Cache store mode defines how the data is stored on the database. This mode can have any one of the three values – **USE**, **BYPASS**, and **REFRESH**.

When the mode is set to **USE** then the data is created or updated when the data is committed to the database. When the mode value is set to **BYPASS** then the value is not written to the cache when it is written to the database. When the mode value is set to **REFRESH** then the data is read from the database, the value in the cache is refreshed.

Slide 33

Let us understand the code used to set the store and retrieval mode of cache.

Specifying the Cache Retrieval and Store Mode 2-3

- Following code snippet demonstrates how to set cache retrieval or store mode:

```
EntityManager em = ...;
em.setProperty("javax.persistence.cache.storeMode",
"BYPASS");
```
- The cache mode can also be set through the **Query** and **TypedQuery** objects through **setHint()** method.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 33

Use slide 33 to explain how to set the store mode of the cache. Developers can use `setProperty()` method of `EntityManager` class or `setHint()` method of `Query` and `TypedQuery` classes.

Slide 34

Let us understand the usage of `setHint()` method to set the cache lock mode.

Specifying the Cache Retrieval and Store Mode

3-3

□ Following code snippet demonstrates setting cache mode through `Query` and `TypedQuery` classes:

```
EntityManager em = ...;
CriteriaQuery<Customer> cq = ...;
TypedQuery<Customer> q = em.createQuery(cq);
q.setHint("javax.persistence.cache.storeMode",
"REFRESH");
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 34



Use slide 34 to demonstrate how developers can set the cache store mode for an entity class using the `setHint()` method of `TypedQuery` class.

Slide 35

Let us understand the Cache interface provided by Java EE.

**Controlling the Cache Settings
Programmatically 1-8**

❑ Developers can use methods defined in `javax.persistence.Cache` to define the caching behavior of the application.

The `Cache` interface has the methods defined for following tasks:

- To check if a given entity has cached data or not
- To remove a particular entity from the cache
- To remove instances of a given entity class from the cache
- To clear cache and remove all entity data from it

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 35

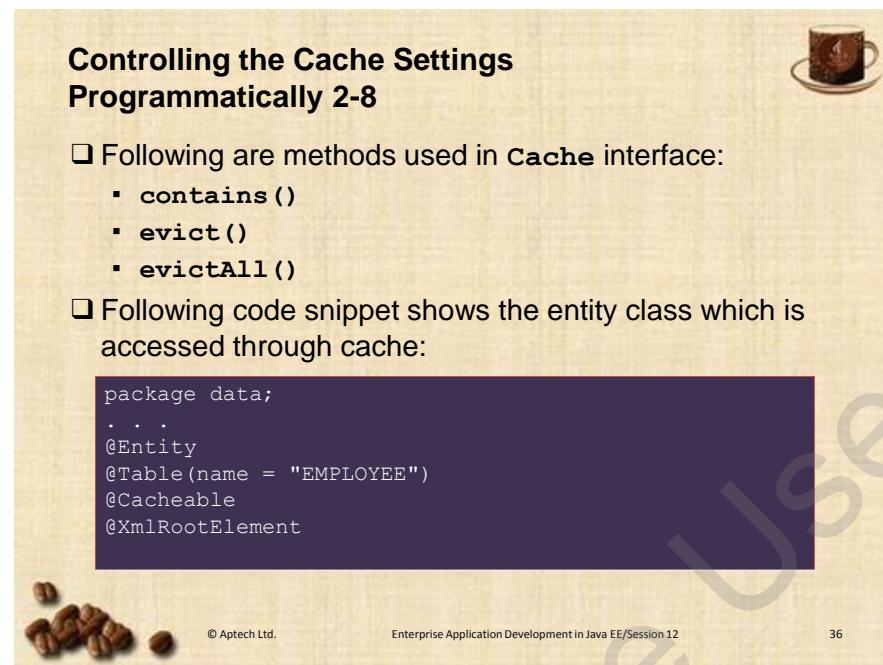
Use slide 35 to explain how developers can use the Cache interface.

Cache interface provided by Java EE platform can perform the following operations:

- It can check if a given entity has cached data or not.
- It removes a particular entity from the cache.
- It removes instances of a given entity class from the cache.
- It clears cache and removes all entity data from it.

Slide 36

Let us understand the methods provided by Cache interface.



Controlling the Cache Settings Programmatically 2-8

Following are methods used in **Cache** interface:

- **contains()**
- **evict()**
- **evictAll()**

Following code snippet shows the entity class which is accessed through cache:

```
package data;
...
@Entity
@Table(name = "EMPLOYEE")
@Cacheable
@XmlElement
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 36

Use slide 36 to explain the methods defined in Cache interface.

Following are the methods defined in Cache interface:

- **contains()** – Accepts the entity class and search attribute as input parameters and returns a boolean value. The returned value signifies whether the given entity is present in the cache or not.
- **evict()** - This method is an overloaded method which accepts an entity class as parameter and removes all the instances of the given entity class from the cache. The overloaded method accepts the entity class name and search attribute as parameters and removes the entity from the cache.
- **evictAll()** – Removes all the entities in the cache and returns a null value.

Slides 37 to 40

Let us look at an example of an entity class which is to be accessed through Cache.

Controlling the Cache Settings Programmatically 3-8

```
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "EMP_ID")
    private Integer empId;
    @Size(max = 15)
    @Column(name = "ENAME")
    private String ename;
    @Column(name = "SALARY")
    private Double salary;
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

37

Controlling the Cache Settings Programmatically 4-8

```
public Employee() {
}

public Employee(Integer empId, String name,
Double sal) {
    this.empId = empId;
    this.ename = name;
    this.salary = sal;
}

public Integer getEmpId() {
    return empId;
}

public void setEmpId(Integer empId) {
    this.empId = empId;
}
```



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

38

Controlling the Cache Settings Programmatically 5-8



```

public String getEname() {
    return ename;
}
public void setEname(String ename) {
    this.ename = ename;
}
public Double getSalary() {
    return salary;
}
public void setSalary(Double salary) {
    this.salary = salary;
}
@Override
public int hashCode() {
    int hash = 0;
    hash += (empId != null ?
empId.hashCode(): 0);
    return hash;
}

```

© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

39

Controlling the Cache Settings Programmatically 6-8



```

@Override
public boolean equals(Object object) {
    /* TODO: Warning - this method won't work in the case
the id fields are not set */
    if (!(object instanceof Employee)) {
        return false;
    }
    Employee other = (Employee) object;
    if ((this.empId == null && other.empId != null) ||
(this.empId != null && !this.empId.equals(other.empId))) {
        return false;
    }
    return true;
}
@Override
public String toString() {
    return "data.Employee[ empId=" + empId + " ]";
}

```

© Aptech Ltd.

Enterprise Application Development in Java EE/Session 12

40

Use slides 37 to 40 to show the code of an Entity class Employee. This entity class is used to demonstrate cache operation when it is accessed with appropriate cache modes set.

Slides 41 and 42

Let us understand the code where the entity class is accessed through Cache.

Controlling the Cache Settings Programmatically 7-8



Following code snippet shows how an entity class is accessed through cache:

```
package access;
...
import javax.persistence.Persistence;
import javax.persistence.PersistenceContext;
@Stateful
@LocalBean
@Cacheable
public class AccessEmployee {
    @PersistenceContext (unitName = "CacheAccessPU")
    public EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("CacheAccessPU");
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 41

Controlling the Cache Settings Programmatically 8-8



```
public EntityManager em;
Employee E = new Employee(101,"Alex",25000D);
Cache C = emf.getCache();
public void store(){
    String out = E.getEname();
    System.out.println("Here "+ out);
    if( C.contains(Employee.class, out))
    {
        System.out.println("In cache");
    }
}
public static void main(String args[]){
    AccessEmployee A = new AccessEmployee();
    A.store();
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 42

Use slides 41 and 42 to explain the code which accesses the entity instances of the entity class through Cache.

The client is accessing a persistence unit CacheAccessPU. The access to the persistence unit is defined cacheable through `@Cacheable` annotation.

The `contains()` method of the Cache class checks whether a given object is in the cache and if it is in the Cache then it executes the following block and prints 'In cache'.

Slide 43

Let us summarize the session.

Summary

- ❑ Concurrent access to the database is always protected with the transaction isolation techniques.
- ❑ Database providers provide locking techniques to maintain data integrity. In addition to that, persistence providers also provide delay database writes, until the end of the transaction.
- ❑ Optimistic locking technique does not acquire locks for providing concurrency control, while performing any operation on certain data.
- ❑ Optimistic locking can be done either by annotating the entity field with javax.persistence.Version annotation or specify the version attribute in the XML descriptor file.
- ❑ Sometimes, it is desirable to acquire the database locks for a long-terms on entities. Such immediate obtained database locks are referred to as 'pessimistic' locks.
- ❑ JPA provides lock modes which are layered on the top of the @Version annotation provided in the optimistic and pessimistic locking.
- ❑ An entity contains a predefined set of lifecycle events that are triggered on the execution of methods called from EntityManager or Query API.
- ❑ The entity listeners are classes that intercept entity callback events.
- ❑ You can specify a set of default entity listeners that are applied to every entity class in the persistence unit.
- ❑ Caching is a technique used by enterprise applications to improve the application performance.
- ❑ First level caching stores the data in the cache for the duration of the transaction or application request.
- ❑ Second level caching can span across multiple transactions and EntityManagers to improve application performance.
- ❑ Developers can use methods defined in javax.persistence.Cache to define the caching behavior of the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 12 43

Using slide 43, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

12.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Java EE Security mechanisms that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 13 – Security

13.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

13.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe enterprise application security
- Explain how to implement security at various levels in an application
- Explain how roles, users, and user groups are defined in an application
- Define authorization and authentication mechanisms used in enterprise applications
- Explain JASS architecture and its services
- Explain how to secure application clients

13.1.2 Teaching Skills

To teach this session successfully, you should be aware of how security mechanisms are defined for enterprise applications and how developers and administrators can define different security roles in the application. You should also be aware of the process of defining authentication and authorization mechanisms for the enterprise applications.

You should prepare yourself with the architecture of Java Authentication and Authorization Service (JAAS) and the services provided by it for applications. Then, implement some examples or code snippets to be taken in the class for securing the enterprise applications from unauthorized access.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

Objectives

- Describe enterprise application security
- Explain how to implement security at various levels in an application
- Explain how roles, users, and user groups are defined in an application
- Define authorization and authentication mechanisms used in enterprise applications
- Explain JASS architecture and its services
- Explain how to secure application clients

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 2

Tell them that they will be introduced to different security mechanisms provided by Java EE and how a developer/administrator can define the application access. Creating different security roles in the application and defining their respective access rights will be explained along with the authentication and authorization mechanisms that can be provided by Java EE.

Tell them the session also explains Java EE provides Java Authentication and Authorization Service (JAAS) to authorize users before accessing different application components. Tell them that this service can be used by the developer to define various roles in the application which can access different application components and perform various operations on it.

13.2 In-Class Explanations

Slide 3

Let us understand the purpose of security mechanisms in enterprise applications.

Introduction 1-2

When an enterprise application is accessed through the Internet or any other open network:

- The users accessing the application components must be appropriately authenticated and authorized, before they can access the services from the application.

All the application components are deployed on the application server:

- Are logically managed through the container who is responsible for providing security services for the components deployed in it.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 3

Use slide 3 to explain when an application requires a security mechanism.

When an application is accessed by users over the Internet or some other public network, the access should be appropriately authenticated and authorized as the application is exposed to several users.

Tell them that in case of enterprise applications, they have various components that need to be protected from unwanted access. When an enterprise application is accessed through the Internet or any other open network, the users accessing the application components must be appropriately authenticated and authorized, before they can access the services from the application.

All the application components are deployed on the application server and are logically managed through the container. The container is responsible for providing security services for the components deployed in it.

The security requirements of the application are defined by the application domain. Based on the requirements, application developers need to define the security policy using mechanisms such as user authentication, password protection of resources, data encryption, and so on.

Slide 4

Let us understand the ways in which the application components can be secured.

Introduction 2-2

- Security is applied to the application components through:
 - Container that implements the security policy defined in the application code.
 - Classes and interfaces to implement the security policy programmatically.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 4

Use slide 4 to explain how the application components can be secured.

Security mechanisms can be defined either through the container or programmatically through the application code.

In Java EE, the security policy is implemented within the application code through annotations and deployment descriptors. As the application components are accessed through the container, the container implements the security policy defined in the application code.

When the application defines security mechanism programmatically it uses various APIs provided by Java EE, their classes, and interfaces are used to define the security mechanism in the application.

Tips:

The EJB container provides system-level security such as transactions and security to the deployed enterprise beans. The EJB framework allows either declarative also called as container-managed or programmatic which is also called as bean-managed security.

In-Class Question:

After you finish explaining the application component security, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



How can you apply security to the application code in Java EE platform?

Answer:

Using annotations or deployment descriptor.

Slides 5 to 10

Let us understand the levels at which the security mechanism has to implemented.

Implementing Security at Various Levels

- ❑ Enterprise application security is implemented at three different levels:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 5

Application Layer Security

- ❑ Implemented by the container.
- ❑ Firewalls can be used at the application layer level to implement the security requirements.
- ❑ Defined both declaratively and programmatically.
- ❑ Declarative security definition is through deployment descriptors and annotations.
- ❑ Programmatic security definition is through interfaces such as EJBContext provided by Java EE.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 6

Transport Layer Security 1-2

- ❑ Refers to the security mechanisms implemented while the application data is transmitted through the network.
- ❑ Based on Point to Point security mechanism ensuring message integrity, authentication, and confidentiality of data transmitted.
- ❑ Uses cryptographic techniques.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 7

Transport Layer Security 2-2

Following are the steps involved in implementing transport layer security:

Client and server agree upon the cryptographic algorithm

The secret key used for communication is exchanged using public key cryptography and certificate based authentication

The agreed upon secret key is used for exchange of data on the network

Transport layer security is unaware of the contents of message being transmitted.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 8

Message Layer Security 1-2

- Security information is bundled along with Simple Object Access Protocol (SOAP) message.
- Security information travels to the destination along with the message.
- Is an end-to-end security.
- When the message with encrypted information is transmitted from the sender, it passes through several intermediate nodes and reaches the destination.
- Encrypted SOAP message is only decrypted by the receiver.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 9

Message Layer Security 2-2

Following figure demonstrates the implementation of security at various levels:

Source → Intermediate node → Intermediate node → Destination

Transport layer security

Application security and Message layer security

Unlike transport layer security, message layer security can be selectively applied on a part of the message.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 10

Use slides 5 to 10 to explain different levels at which the security mechanisms can be implemented in the application.

Application security has to be implemented at three levels – Message level security, Transport layer security, and Application layer security.

At each of these three levels the application would implement different security mechanisms such as implementing cryptographic techniques, defining authentication and authorization mechanism, and so on to achieve the desired effect.

Use slide 6 to explain application layer security.

Application security defines security policy for the entire application. The security policy for all the application components is defined at this level. The policy is defined by the container. Apart from the programmatic methods, applications can configure firewalls for the application to implement application level security.

The security definition can be done either programmatically or declaratively. When done declaratively the required mechanism is specified through deployment descriptors and annotations.

In order to programmatically define the security mechanism, developers can use classes and interfaces provided by Java EE.

Use slides 7 and 8 to explain the security mechanisms which are implemented at the transport layer level. Transport layer security refers to the mechanisms which are defined when application data traverses through the network.

These techniques are required to ensure the integrity of data and also ensure that the data is not disclosed to unwanted users during traversal. Applications use cryptographic techniques at this stage.

The steps involved in implementing transport layer are that both the sender and receiver should agree upon the cryptographic algorithm. Based on the algorithm they agreed upon they should exchange keys.

Use slide 9 to explain message layer security in the application. Message layer security is based upon Simple Object Access Protocol (SOAP). The security information travels to the destination along with the message. This is also known as end-to-end security.

Message layer security also uses cryptographic techniques. Message layer security can be selectively applied on a part of the message. It can be implemented independent of application environment.

Use slide 10 to explain the scope of each security layer or level mechanism implemented in the application with the help of the figure as shown on slide 10. Tell them that Application level security and message level security is applied from the source to the destination. Transport layer security is applied between adjacent nodes in network traversal. The message level security can be implemented on the entire message or only part of the application message.

Tips:

Cryptography is a field of study where several mathematical algorithms are used to transform data into unreadable format when the data is transmitted from the source. The data traverses in unintelligible form, at the receiver end the data is transformed into readable format using mathematical formulae.

There are two types of cryptographic algorithms available – symmetric cryptographic algorithm and asymmetric algorithms. In case of symmetric algorithm same key is used by the sender and receiver to encrypt and decrypt data, whereas in asymmetric algorithm the sender and receiver can use different algorithms for encryption and decryption.

Slide 11

Let us understand the characteristics of security mechanisms.

Characteristics of Security Mechanisms

- ❑ Prevent unauthorized access to application data and components.
- ❑ Identity of an application user should be associated with each action performed on the enterprise application.
- ❑ Users cannot deny the operations performed.
- ❑ Protects the application from service failures such as server crash, network failure, and other interruptions.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 11

Use slide 11 to explain the characteristics of a security mechanism.

A security mechanism has to implement certain characteristics in the application. Following are the characteristics of a security mechanism:

- Prevent unauthorized access to the application data and application components or resources.
- When an application user performs certain operations on the enterprise application, then the identity of the user should be associated with the operations performed. The user cannot deny the performed operations. This characteristic is essential to trace the users who have performed malicious operations in the application.
- Protect the application from service failures such as server crash, network failure, and other interruptions.

Tell them, that the properly implemented security mechanism also allows:

- The proper administration of the application and its components
- Transparency to system users

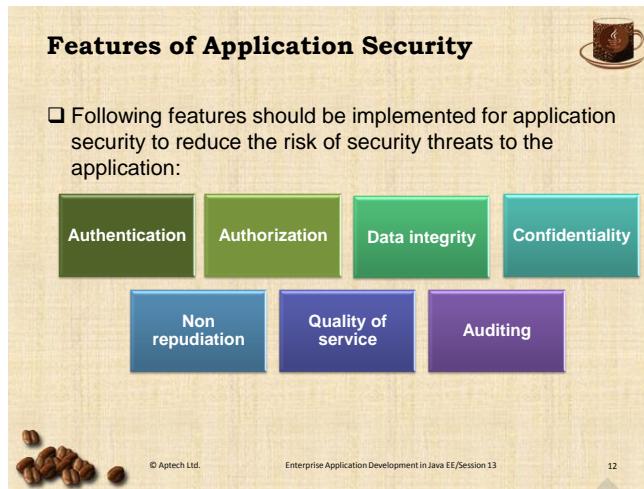
Additional References:

To get more information on security mechanisms, refer the following link:

<http://docs.oracle.com/cd/E19798-01/821-1841/bnwy/index.html>.

Slide 12

Let us understand the features of application security.



Use slide 12 to explain the features of application security.

Following are the required features of an application security mechanism:

Authentication is a process by which the identity of the user is determined. The most common form of authentication involves the use of username and password.

Authorization controls what the authenticated user is allowed to do after he/she is granted access.

Data integrity is a characteristic which requires that the information is not modified by unwarranted users. Applications implement various checks on the data such as Cyclic Redundancy Checksum (CRC) codes and so on to verify the integrity of data.

Confidentiality implies secrecy, where by the security system allows access of data only to authorized users of the application.

Quality of Service has to be achieved such that this additional overhead does not affect the application performance.

Auditing of an application log is done to ensure that the application is performing as expected.

Tips:

Issuing digital certificates to the users requires a genuine Certification Authority (CA) to be implemented in the application domain.

Slides 13 and 14

Let us understand more about authentication.

Authentication 1-2



- Is the process by which one entity in an interaction determines the identity of the other.
- In Java EE environment:
 - The EJB server determines the identity of all types of clients so, that it can determine the level of access to be granted.
 - The client may also want to authenticate the server, to ensure that it is interacting with the correct server.
- The most common form of authentication involves the use of username and password.
- The use of digital certificates offers a stronger form of authentication.

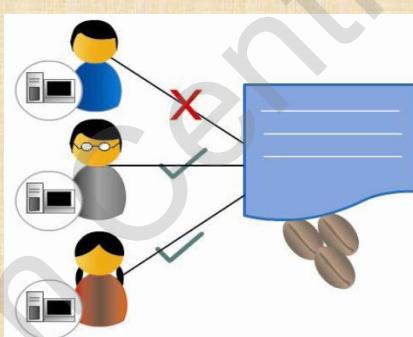


© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 13

Authentication 2-2



- Following figure depicts authentication:



© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 14

Use slides 13 and 14 to explain authentication.

The security mechanism has to allow the server to authenticate the client entering in the application. The most common method implemented for this authentication is prompting for the username and password.

In an EJB application, clients of EJBs may be applications or other EJBs. The EJB server determines the identity of all types of clients so, that it can determine the level of access to be granted. On the other hand, the client may also want to authenticate the server, to ensure that it is interacting with the correct server.

There are different variations of this scheme, which involves different methods for encrypting and transmitting the password. One main concern with password authentication is that if someone else fraudulently gets your username and password, they can assume your identity.

The use of digital certificates offers a stronger form of authentication. Digital certificates can be used to identify end-users, servers, and other software components. A digital certificate is an electronic

identity card that establishes your credentials. It is issued by a third-party Certification Authority (CA) and contains a serial number, expiration dates, the certificate holder's public key, and the digital signature of the CA. The certificate holder's public key is used for encrypting messages and the digital signature of the certification authority verifies that the certificate is real.

Digital certificate-based authentication can be used only on the server, the client, or both, depending on the needs of the application. In most of the cases, mutual digital certificate based authentication is used so that both the client and the server are confident of each other's identity.

Use slide 14 to explain the authentication process graphically where one of the users is not granted access to the EJBs based on the credentials provided.

Slides 15 and 16

Let us understand more about authorization.

Authorization 1-2

- ❑ In an enterprise application, client authentication is usually followed by authorization.
- ❑ **Authentication** - Ensures only valid users get access to the application.
- ❑ **Authorization** - Controls what the authenticated user is allowed to do after he/she is granted access.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 15

Authorization 2-2

- ❑ Following figure depicts authorization:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 16

Use slides 15 and 16 to explain the authorization characteristic of the security mechanism.

Authorization refers to the mechanism of granting access rights to an application user. Every user of the application has a set of valid operations which he/she can perform. Authorization process defines the user roles and their respective access rights to be granted.

Slide 17

Let us understand more about data integrity.

Data Integrity

- ❑ Data integrity is a characteristic which requires that the information is not modified by unwarranted users.
- ❑ Applications implement various checks on the data such as Cyclic Redundancy Checksum (CRC) codes and so on to detect whether the information is modified by any third party users.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 17

Use slide 17 to explain data integrity.

The communication between the client and server should not be modified by the unwarranted users. To do so, applications use Cyclic Redundancy Checks to detect whether the information is modified by any third party users.

Tips:

Cyclic Redundancy Check is a mechanism which generates a code based on the message transmitted over the network. The sender generates this code and sends it along with the message transmitted. The receiver of the message recalculates the code and matches it with the received code. If there is a discrepancy in the codes matched then it implies that the message is distorted.

In-Class Question:

After you finish explaining the data integrity, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the purpose of authorizing the users in the enterprise applications?

Answer:

Authorization refers to the mechanism of granting access rights to an application user.

Slide 18

Let us understand confidentiality and non-repudiation.



Confidentiality and Non-repudiation

- ❑ Confidentiality implies secrecy, where by the security system allows access of data only to authorized users of the application.
- ❑ Non-repudiation is the security mechanism that associates the identity of the user with actions performed by them on the application.
- ❑ If a user performs a malicious operation on the application, the security mechanism ensures that the user does not deny the operations performed.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 18

Use slide 18 to explain the confidentiality and non-repudiation features of security mechanism.

Confidentiality requirement of the application states that the application data has to be revealed only to authorized users of the application.

Non-repudiation characteristic of the application implies that the user identity has to be associated with each action performed in the application. The users who have performed an action cannot deny their association with the action. These entries are made in the application log.

Slide 19

Let us understand Quality of Service and Application Auditing.



Quality of Service and Auditing

- ❑ The security mechanism implemented in the application increases the application execution time.
- ❑ For instance, when access to a resource requires username and password. The application execution cannot proceed until the user provides the appropriate information.
- ❑ Auditing of an application log is done to ensure that the application is performing as expected.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 19

Use slide 19 to explain quality of service and auditing characteristics of the security mechanism.

Quality of service implies that inspite of implementing all the security mechanisms, the application execution time should not significantly increase to reduce the application performance.

All the operations performed in the application should be properly logged. Application logging and auditing are helpful in system recovery in case of application failure and also in assessing the performance of the application.

Slide 20

Let us understand a simple security mechanism.

**Simple Application Security Implementation
1-3**

Following are the steps to be performed to implement the security requirements of the application:

- Every user is provided with a unique username.
- Each username is linked with the account held by the user.
- Users should be authenticated to access their account.
- Authentication should be followed by authorizing the user.
- The operations of checking the account balance and transfer funds are to be implemented in each account.
- Enterprise beans should be invoked to perform the required operations.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 20

Use slide 20 to explain a simple security mechanism used in the application.

Consider the context of a bank application with following requirements:

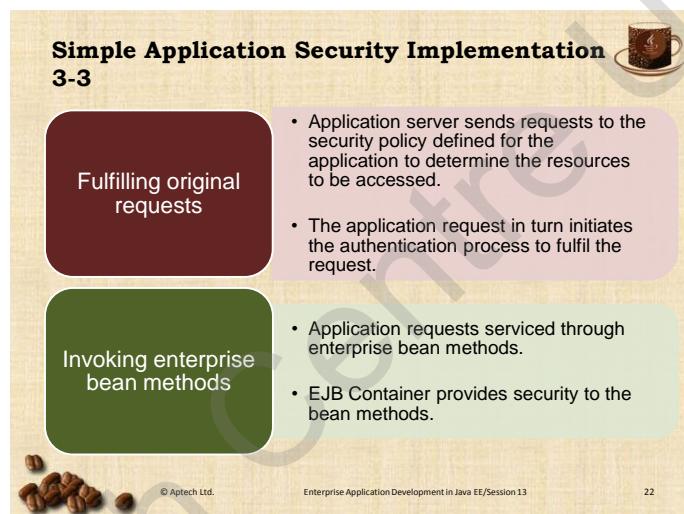
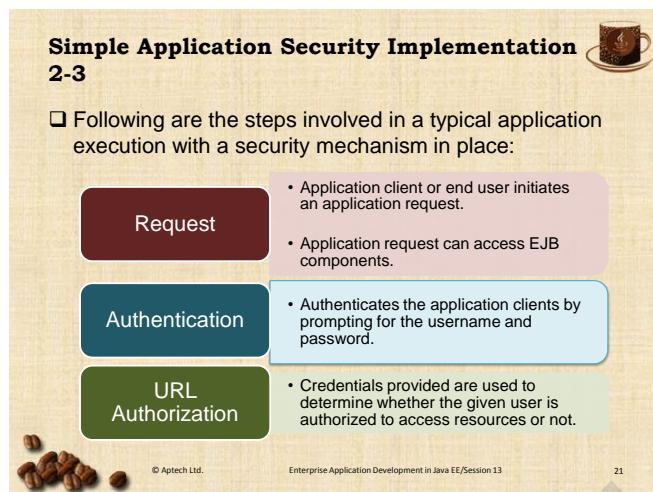
- Every customer of the bank is provided with online access to the account.
- The customer can only access his/her account. The customer can check balance in his/ her account, transfer funds from the account to another account through the online portal.

To implement the required security mechanism the application has to implement the following operations:

1. Every user is provided with a unique username. This username is linked with the account held by the user.
2. Whenever a user tries to access an account, the application server should prompt for a user name and a corresponding password.
3. The application server checks the username and password against the customer database it holds and verifies the information. This process is known as authentication.
4. The customer of a bank can perform operations such as check the account balance and transfer funds to other accounts. Therefore, the username of the customer is associated with the operations the user can perform.
5. This process of allowing a user to perform certain operations is known as authorization. Usually the authentication process is followed by the authorization process.
6. Once the user authentication and authorization is complete, the customer can perform the necessary operations such as funds transfer and so on. The user need not enter the username and password for every operation performed, the user authorization is valid for a certain time period.
7. To perform the operations initiated by the customer, the application has to invoke enterprise beans on the application server.

Slides 21 and 22

Let us further refine the requirements of the application.



Use slides 21 and 22 to further refine the requirements of the security mechanism.

- Request** - The application client or end user initiates an application request, this application request may have to access EJB components deployed in the container.
- Authentication** - The application server authenticates the application clients by prompting for the username and password. The application server may also use any other security mechanism to validate the user.
- URL authorization** – The credentials provided are used to determine whether the given user is authorized to access resources or not. Access to appropriate resources is provided according to the security policy of the application.
- Fulfilling original requests** - The application server sends requests to the security policy defined for the application to determine the resources to be accessed. The application request which in turn has initiated the authentication process is then fulfilled.
- Invoking Enterprise Bean methods** – The application request from the client is satisfied by invoking appropriate enterprise bean methods. The EJB container is responsible for securing the enterprise bean methods. The container refers to the security policy of the application and invokes the bean methods to which the user has access rights.

Slides 23 and 24

Let us understand Access Control Lists.

Access Control Lists (ACLs) 1-2



❑ Permissions represent a right to access a particular resource or to perform some action on an application.

❑ An administrator:

- Usually protects resources by creating lists of users and groups that have the permission to access a particular resource.
- Lists are referred as Access Control Lists (ACLs).

❑ An ACL file is made up of entries, which contain a set of permissions for a particular resource and a set of users who can access those resources.



© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 23

Access Control Lists (ACLs) 2-2



❑ Following figure shows Access Control Lists:

	Read	Write	Execute
Admin	✓	✓	✓
Professor	✓	✓	✓
Staff	✓		
Student	✓		



© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 24

Use slide 23 to explain Access Control Lists (ACLs) used in security mechanisms. ACLs are used for defining different users of the application and their corresponding access rights.

Each user of the application should have certain permissions to access the resources of the application. Permissions represent a right to access a particular resource or to perform some action on an application. These permissions are listed out in ACLs.

For example, a user with admin permissions on a chat site may create, modify, or close a chat room, but a user with normal permissions may be allowed only to enter a chat room and participate in a chat session.

An ACL file is made up of entries, which contain a set of permissions for a particular resource and a set of users who can access those resources.

Use slide 24 to explain an example of ACL. The application as shown on the slide has four roles. Admin, Professor, Staff, and Student.

The roles Admin and Professor have read, write, and execute rights on the application whereas Staff and Student roles have only read permission while accessing the application.

Slide 25

Let us understand the concepts of Users, Realms, and Groups.

Users, Groups, Roles, and Realms 1-2

- ❑ An application domain has various end users.
- ❑ The users are logically grouped into user groups based on some common characteristics.
- ❑ Every end user is termed as a user of the application; however, different users may have different roles to play in the application.
- ❑ For instance, an employee's role in a bank application is different from the customer's role in the same domain.
- ❑ A realm is a single authentication policy that controls a set of users or user groups.
- ❑ An admin realm in an application therefore, grants administrative rights required by the application to the admin group.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 25

Use slide 25 to explain Users, Groups, Roles, and Realms.

After the application is deployed on the infrastructure, anyone accessing the application is a 'User' of the application. All users are grouped according to the role they play in the application.

Each role is associated with a set of access rights to the application. Users who have similar access rights are grouped into user groups.

Users are given access to application based on their access rights. Before giving access to users they are authenticated, based on their credentials they are provided access to set of resources.

All users who are controlled through same authentication policy form an application realm. A realm is a single authentication policy that controls a set of users or user groups. An admin realm in an application therefore, grants administrative rights required by the application to the admin group.

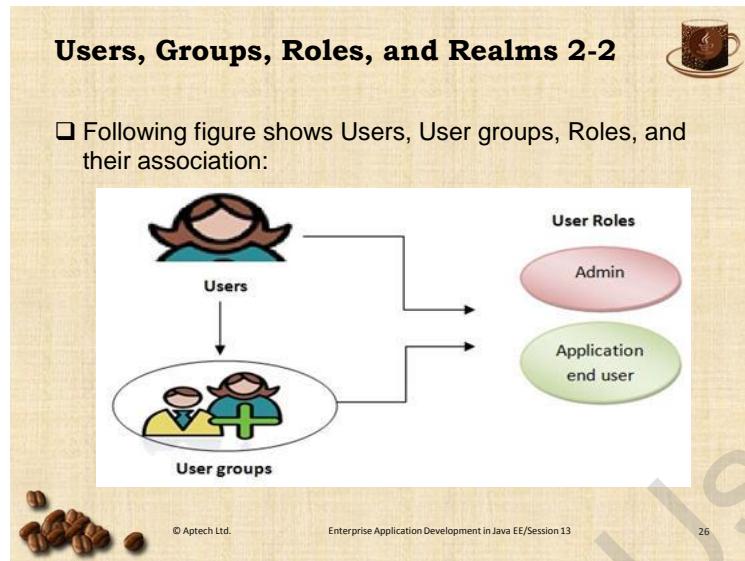
Tips:

J2EE security architecture is based on authorization model, as it is concerned with granting or denial of access to clients that have already been authenticated by the container. This model enables the application developer to concentrate on the needs of the application and leave the technicalities of security and authentication on to the administrator.

The authentication and authorization information and the enforcement policies, as defined by an administrator are referred as realms. Realms are typically repositories containing users, groups, permissions, and secured resources. In other words, the realm contains the usernames and passwords and the authentication and authorization policy details that control these users. A realm can be stored in an XML file, a text file, and sometimes even in a database.

Slide 26

Let us understand users, roles, and user groups.



Use slide 26 to explain users, user groups, and roles.

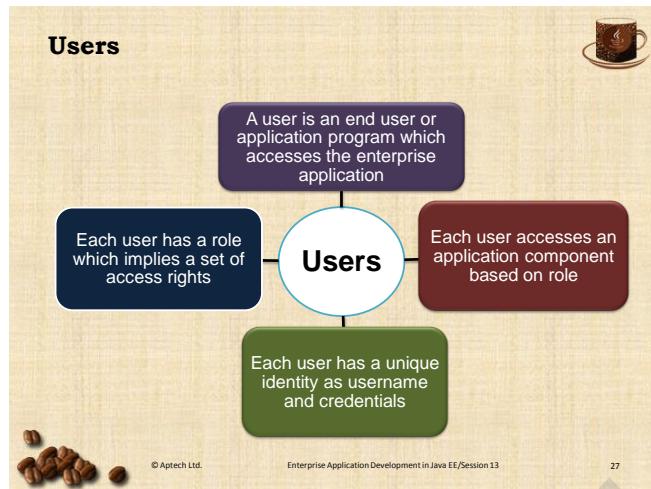
Each user is identified through their credentials. User can have different access rights on the application. Based on the access rights they have, the user is assigned a role in the application domain.

When there are multiple users having same access rights they are grouped as user groups.

User roles represent different sets of access rights. The tasks that can be performed by a user who is in the role of an admin are different from the tasks that can be performed by an application end user. The access rights given to each of these users are different. An application role can be assumed by a single user or a group of users in the application domain.

Slide 27

Let us understand more about users in the application.



Use slide 27 to explain users in the application.

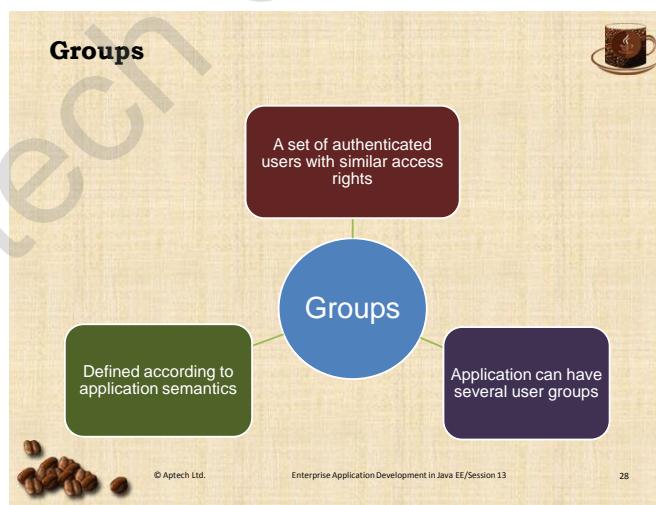
A user is an end user or application program which accesses the enterprise application. The identity of the user is defined on the application server such as GlassFish.

User presents his/her identity which is the username to the application server while trying to access the application resources.

There are no two identical users in the application domain. Each user has its own identity with credentials.

Slide 28

Let us understand more about user groups.



Use slide 28 to explain user groups in the application.

Group refers to a set of authenticated users with similar roles and access rights in the application. These groups can also be defined according to application semantics.

Each application can have several user groups.

Slides 29 and 30

Let us understand roles in the application domain.

Roles 1-2

- ❑ Role in the application domain reflects the set of access rights held by a certain user.
- ❑ For instance, all employees in the bank application do not have the right to credit interest into the accounts of customers.
- ❑ The application has to define a role with the required privileges assigned to that role so that an employee can perform the task of crediting interest into the accounts of the customers.
- ❑ Each role contains a particular set of permissions.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 29

Roles 2-2

- ❑ Following figure shows the groups and roles:

The diagram illustrates the concept of user groups. It shows two distinct groups: 'Role: Manager' and 'Role: Operator'. Each group is represented by a blue rounded rectangle containing three stylized human figures. A single line connects both groups to a central blue rectangular box labeled 'Application Component'. This visualizes how multiple users within defined groups can access the same application components based on their assigned roles.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 30

Use slides 29 and 30 to explain roles in the application domain.

Role in the application domain reflects the set of access rights held by a certain user. Each role contains a particular set of permissions. While creating an EJB, the developer can specify the actions that can be performed by the application users.

Use slide 30 to explain the concept of user groups in the application. The given figure on slide 30 shows two different user groups which can access the application beans. The user groups Manager and Operator have different access rights to access the application components.

Tips:

From security perspective, groups are used primarily to manage multiple users in an efficient manner. When a group is given some access rights and permissions, all members of the group gain this permission indirectly.

Roles are abstract forms of groups. A role is a particular way that a user may interact with an application and it also defines the access rights that the user must have to perform this interaction. For example, manager, administrators, and buyers are roles that are supported in an e-shopping Web site. Each role contains a particular set of permissions. While creating an EJB, you can specify which role can perform what action in your application. You cannot however associate users with the roles supported by your applications. The association of individual users with the users can only be done by the administrator.

Slides 31 and 32

Let us understand the concept of realms.

Realms 1-2

- ❑ Each application has a set of protected resources.
- ❑ Each realm is associated with an authentication scheme to access certain protected resources.
- ❑ Realm is the set of authorized users who can access those protected resources.
- ❑ Java EE supports three default realms:
 - Admin realm
 - Certificate realm
 - File realm

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 31

Realms 2-2

Admin realm	• Stores all the credential data in admin-keyfile. • Authenticates information locally stored.
Certificate realm	• Stores all the credential data in certificate database. • Uses X.509 certificates for user authentication.
File realm	• Stores all the user credentials in keyfile. • Can be used for all the enterprise clients but not for Web browser clients and those using HTTPs..

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 32

Use slides 31 and 32 to explain the concept of realms in the application.

Realm refers to a set of users who are governed by an authentication policy. The protected resources on the application server are partitioned into a set of protection spaces. Each protection space is associated with an authentication scheme and an authorized database of users.

There are three default realms in Java EE server authentication service through which users are governed – **admin realm**, **certificate realm**, and **file realm**.

admin-realm stores all the user credential data in the **admin-keyfile**. The users in the **admin realm** are authenticated with the help of **admin-keyfile**. The authentication information in this case is locally stored.

certificate-realm stores all the user credential data in a certificate database. This mechanism uses X.509 digital certificates for user authentication. While using **certificate realm**, the application server uses **HTTPs** to authenticate the Web client. This mechanism is primarily used for Web applications.

When the application uses **file realm**, user credential data is stored in a file called **keyfile**. The user is authenticated using the data in the **keyfile**.

Additional References:

To get more information on working with Realms, Users, Groups, and Roles, refer the following link:
<http://docs.oracle.com/cd/E19798-01/821-1841/bnbxi/index.html>.

In-Class Question:

After you finish explaining the users, roles, and realms, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



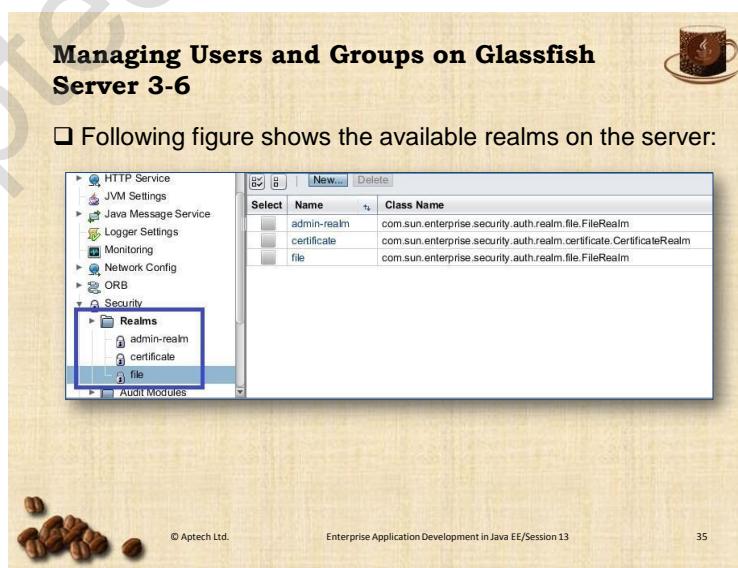
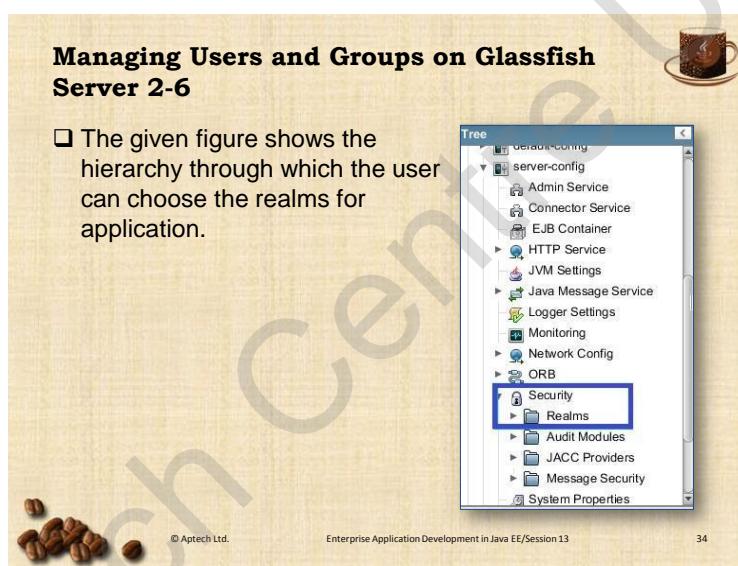
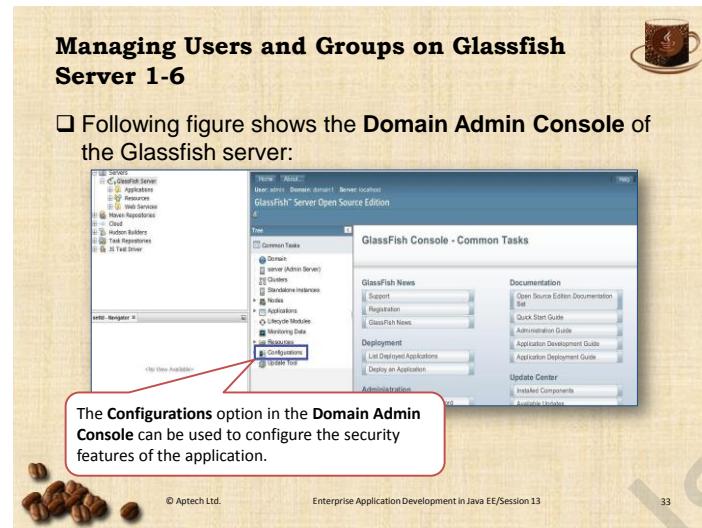
What is the use of creating user groups?

Answer:

Group refers to a set of authenticated users with similar roles and access rights in the application.

Slides 33 to 35

Let us understand managing users, groups, and roles on GlassFish server through NetBeans IDE.



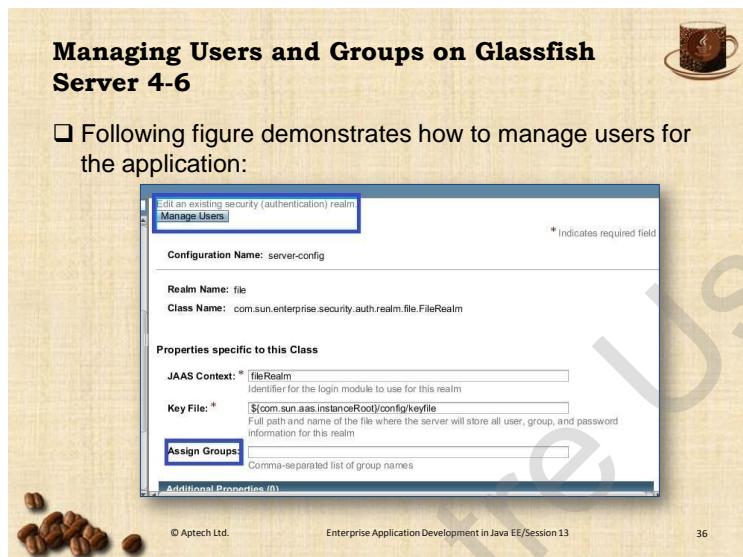
Use slides 33 to 35 to explain how to define users, user groups on GlassFish server.

Select the GlassFish server among the servers in NetBeans IDE. After selecting the GlassFish server, select Configurations tab in the console. Demonstrate where the developer can choose Realms in the 'Configurations' tab. Show the three realms available in the application.

The navigation path in the GlassFish server console is Configuration → Realms.

Slide 36

Let us understand how to manage users in realms.

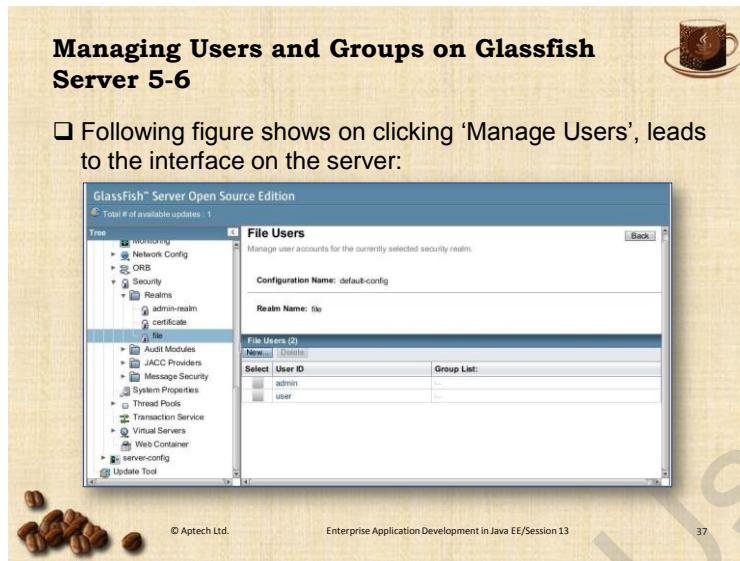


Use slide 36 to demonstrate how users can be managed in different realms.

Click 'Manage Users' to manage users in the realm.

Slide 37

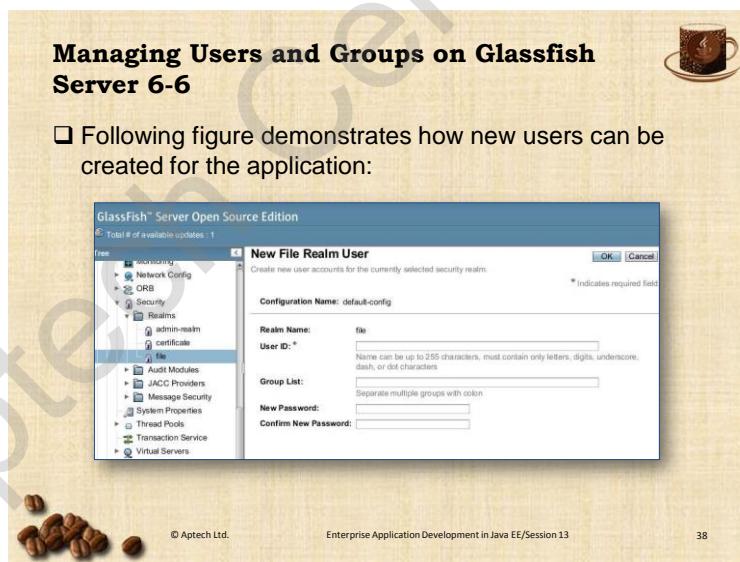
Let us understand how to manage users on the application server.



Use slide 37 to explain user management on the GlassFish server. Here, users are being managed in a file realm. There are two users already defined in this realm – admin and user. In order to add a new user to the realm, click ‘New’ button to add in the list of users.

Slide 38

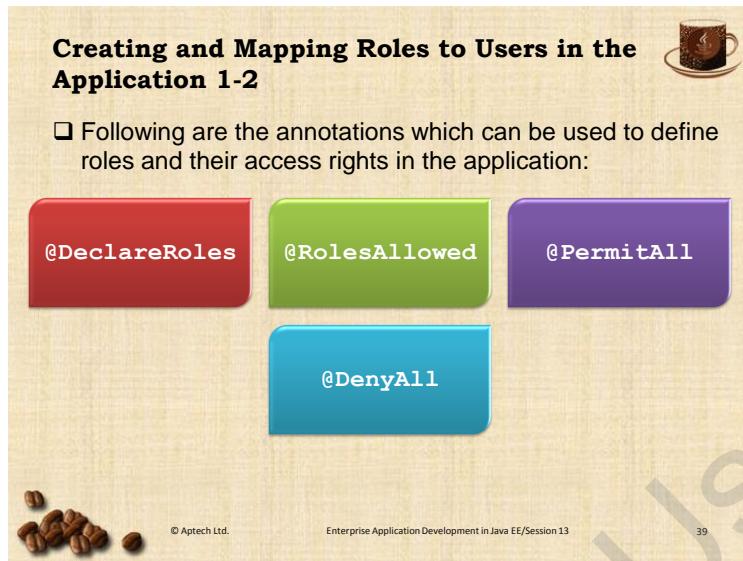
Let us understand the process for adding a new user in the realm.



Use slide 38 to explain the process of creating new user in the realm. In the given figure as shown on the slide developer can add the username and password details of the user.

Slide 39

Let us understand the annotations that can be used to define users and roles in the application.



Use slide 39 to explain different annotations which can be used to define users and roles in the application.

Tell them that it make the deployer's task easier, the application developer can define security roles.

Security Role

A security role is a grouping of permissions that a given type of application users must have in order to successfully use the application. For example, in a payroll application, some users will want to view their own payroll information (employee), some will need to view others' payroll information (manager), and some will need to be able to change others' payroll information (payrollDept). The application developer would determine the potential users of the application and which methods would be accessible to which users. The application developer would then decorate classes or methods of the enterprise bean with annotations that specify the types of users authorized to access those methods.

Some of the annotations that can be used to define security roles are as follows:

- `@DeclareRoles` annotation is used to declare roles in the application domain which have the right to access the application resources or business methods.
- `@RolesAllowed` annotation is used to specify the application roles that are allowed to access the application resource.
- `@PermitAll` annotation specifies that all the security roles can access the given bean method. This does not invoke any authentication mechanism. This annotation can be used for both bean classes and bean methods.
- `@DenyAll` annotation specifies that no security role can access the current bean class or bean method. Such methods are excluded from execution in Java EE container.

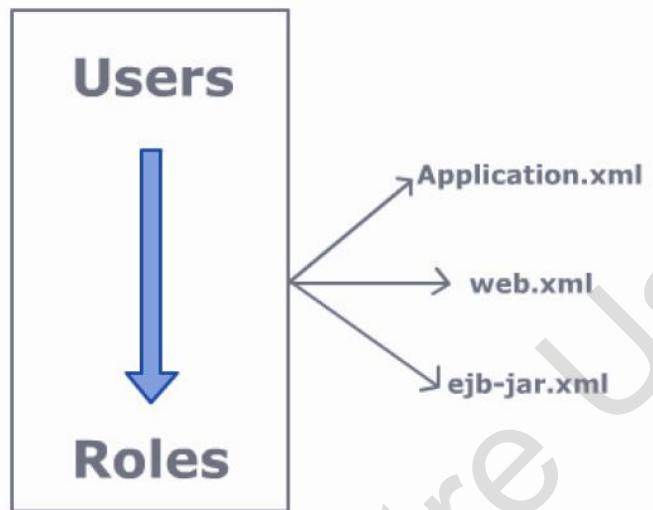
Additional References:

To get more information about on specifying authorized users by declaring security roles, refer the following link: <http://docs.oracle.com/cd/E19798-01/821-1841/gjgcq/index.html>.

Tips:

For enterprise applications, you define security roles in the deployment descriptor file `application.xml`.

For individually deployed EJB modules, you define roles in the EJB's deployment descriptor `ejb-jar.xml` and similarly, you define the security roles for the Web application in `web.xml`. The following figure depicts security roles:

**Slide 40**

Let us understand the usage of annotations in demonstrating the roles in the application.

Creating and Mapping Roles to Users in the Application 2-2

Following code snippet shows the usage of `@DeclareRoles` and `@RolesAllowed`:

```

import javax.annotation.security.DeclareRoles;
import javax.annotation.security.RolesAllowed;
...
@DeclareRoles({"VALUATOR", "MANAGER"})
@Stateless public class LoanApprovalBean {
    @Resource SessionContext ctx;
    @RolesAllowed("VALUATOR")
    public void reviewPropertyValue(PropertyInfo info) {
        ...
    }
    @RolesAllowed("MANAGER")
    public void ApproveLoan(PropertyInfo info) {
        ...
    }
    ...
}
  
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 40

Use slide 40 to explain how the developer can define the roles in the application.

The code as shown on the slide declares two roles `VALUATOR` and `MANAGER` who can access the class `LoanApprovalBean`.

The method `reviewPropertyValue()` can be accessed by the role `VALUATOR` in the application.

The method `approveLoan()` can be accessed by the role `MANAGER`.

Slide 41

Let us understand the process of establishing a secure connection.

Establishing a Secure SSL Connection

- ❑ Secure Sockets Layer (SSL) is used at transport layer for secure communication.
- ❑ Uses cryptographic techniques.
- ❑ Implements point-to-point security.
- ❑ Implements three important characteristics of security mechanism:
 - Authentication
 - Confidentiality
 - Integrity

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 41

Use slide 41 to explain the process of establishing a secure SSL connection.

The transport layer security of the application is implemented through Secure Sockets Layer (SSL).

SSL implements various cryptographic techniques to implement point-to-point security.

Two communicating entities agree upon a shared secret key using public key cryptography. The communicating entities then use the shared secret key to encrypt the data at the source and decrypt the data at the destination. Though SSL is efficiently designed, the cryptographic process adds an overhead on application performance.

Slides 42 and 43

Let us understand the different roles in an enterprise application.

Security Tasks in Enterprise Applications 1-2



□ Application security is implemented by:

- **System administrators** - responsible for creating roles and users.
- **Application developers** - provides access rights to different roles using annotations or deployment descriptors.
- **Application deployers** - responsible for deploying the application on the server according to the security specifications provided in the deployment descriptor.
- **Bean providers** - supports the security mechanisms required by the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 42

Security Tasks in Enterprise Applications 2-2



□ Following are the tasks that must be performed as part of security implementation for applications:

1. Creating a database of users who will be accessing the application.
2. Defining relevant user groups according to the application context.
3. Assigning the users to appropriate groups.
4. Propagating the user identity across all the application components.
5. Configuring the application server with appropriate user and role mappings.
6. Annotating the classes appropriately to declare roles and defining the access to be granted to different roles.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 43

Use slides 42 and 43 to explain different roles associated with implementing security tasks in the application.

System administrators, application developers, application deployers and bean providers are different roles associated with defining security mechanisms in application development.

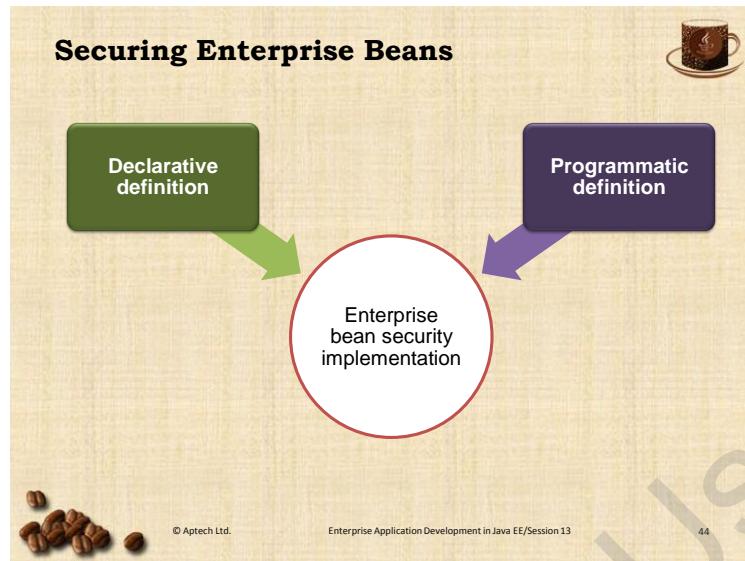
- System administrators are responsible for creating roles and users.
- Administrators are responsible for mapping the users onto appropriate roles and user groups.
- Developers can create roles and provide access rights to different roles using annotations or deployment descriptors.
- Deployers are responsible for deploying the application on the server according to the security specifications provided in the deployment descriptor.
- The bean provider supports the security mechanisms required by the application.

Use slide 43 to explain different tasks which are to be implemented as part of the security implementation of enterprise applications.

1. Creating a database of users who will be accessing the application.
2. Defining relevant user groups according to the application context.
3. Assigning the users to appropriate groups.
4. When the user provides authentication information, propagating the data across all the application components. This process is known as identity propagation, where the authenticated user ID is propagated across all relevant application components.
5. Configuring the application server with appropriate user and role mappings.
6. Annotating the classes appropriately to declare roles and defining the access to be granted to different roles.

Slide 44

Let us understand different methods of implementing security in enterprise beans.



Use slide 44 to explain the two methods based on which security mechanisms for enterprise beans can be defined.

Security mechanisms on enterprise beans can be defined declaratively or programmatically.

Declarative definition is through deployment descriptors and annotations. The presence of an annotation in the business method of an enterprise bean class specifies method permissions is all that is needed for method protection and authentication in some situations.

Programmatic definition of security mechanisms can be done through classes and interfaces. For an enterprise bean, the code is embedded in a business method that is used to access a caller's identity programmatically and that uses this information to make security decisions.

Programmatic security is useful when declarative security alone is not sufficient to express the security model of an application.

Slide 45

Let us understand the definition of declarative security mechanism.

**Securing an Enterprise Bean Method
Declaratively**

- ❑ The application deployer defines the security features on the application server based on the deployment descriptor and annotations.
- ❑ The deployer defines the users, user groups, and their respective roles on the application server.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 45

Use slide 45 to define declarative security mechanism. Applications implement declarative security mechanism using deployment descriptors and annotations. The deployer defines the users, user groups, and their respective roles on the application server.

Additional References:

To prepare an example on securing enterprise applications using declarative approach, refer the following link: <http://docs.oracle.com/cd/E19798-01/821-1841/gigdi/index.html>.

Slides 46 to 48

Let us understand defining security mechanism programmatically.

**Securing an Enterprise Bean Method
Programmatically**

- ❑ In this method of specifying the security mechanism, the developer uses the security APIs and methods to define the security mechanisms.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 46

Accessing an Enterprise Bean Caller's Security Context 1-2



© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 47

- ❑ javax.ejb.EJBContext provides methods to access security information about the user or entity who is invoking the enterprise bean method.
- ❑ Following are the methods provided by the EJBContext interface:
 - getCallerPrincipal()
 - isCallerInRole()

Accessing an Enterprise Bean Caller's Security Context 2-2



© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 48

- ❑ Following code snippet demonstrates the usage of isCallerInRole() method:

```
.....
@Resource Session Context X
if(X.isCallerRole(admin)==true)
{
    System.out.println("Admin right assigned");
}
else
    System.out.println(" No Admin rights");
....
```

Use slides 46 to 48 to explain specifying security mechanisms programmatically.

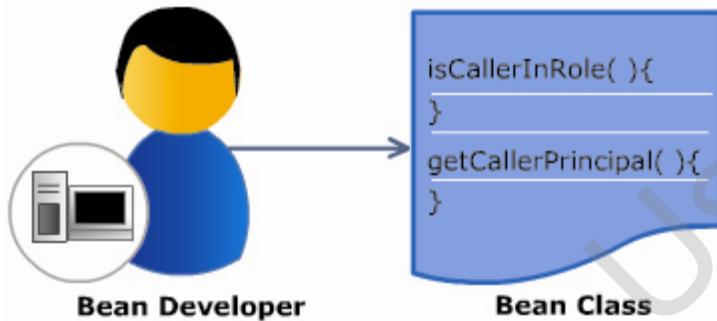
- Java EE provides various APIs to define application security mechanisms. The EJBContext interface provides various methods to implement security among the applications. The javax.ejb.EJBContext - provides methods to access security information about the user or entity who is invoking the enterprise bean method. Following are the methods provided by EJBContext class.

Use slide 47 to explain the classes and methods used by enterprise beans to define security mechanisms in the application.

- getCallerPrincipal() - This method is used to retrieve the name of the invoking entity or the user. The security methods may then use the name of the entities to check the user database to determine the role of the current entity.

- `isCallerInRole()` –
This method is used to check whether the current user has been assigned with certain role or not. The application developer defines the roles for the users. If an application context object 'X' is trying to access methods which are meant for administrator, then the security mechanism can check whether the object has administrative rights or not through `isCallerInRole()` method.

Following figure shows the programmatic authorization:



Use slide 48 to explain the usage of `isCallerInRole()` method. The method in the code checks whether the user who is accessing the Context X in the application is an admin or not.

If the user is an admin then execute the first block of `if()` statement else the second block of `if()` statement.

Tips:

Following is the additional code snippet that illustrates the use of the `isCallerInRole()` method:

```

//Other EJB code here
EJBContext ctx;
//...
public void makePurchase(Integer productId, double newAmount, String
userNames) throws InvalidPurchaseException{

// Make sure this user is registered
if (!ctx.isCallerInRole("registered_user")) {
    throw new InvalidPurchaseException("You must register first");
}
...

```

The `isCallerInRole()` method is called on the EJB context to check whether the current user belongs to `registered_user` role or not. If the user does not belong to `registered_user` role, an exception is thrown.

Following code snippet illustrates the use of the `getCallerPrincipal()` method:

```
//Other EJB code here  
EJBContext ctx;  
//...  
Principal principal = ctx.getCallerPrincipal();  
String buyerName = null;  
if (principal != null) {  
    buyerName = principal.getName();  
} else {  
    buyerName = "NA";  
}
```

The `getCallerPrincipal()` method is called on the EJB context to retrieve the `Principal` object, and then the name of the `Principal` is obtained by calling the `getName()` method.

Additional References:

To prepare an example on securing the enterprise application programmatically, refer the following link: <http://docs.oracle.com/cd/E19798-01/821-1841/gjgcs/index.html>.

In-Class Question:

After you finish explaining how to specify security mechanisms programmatically, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



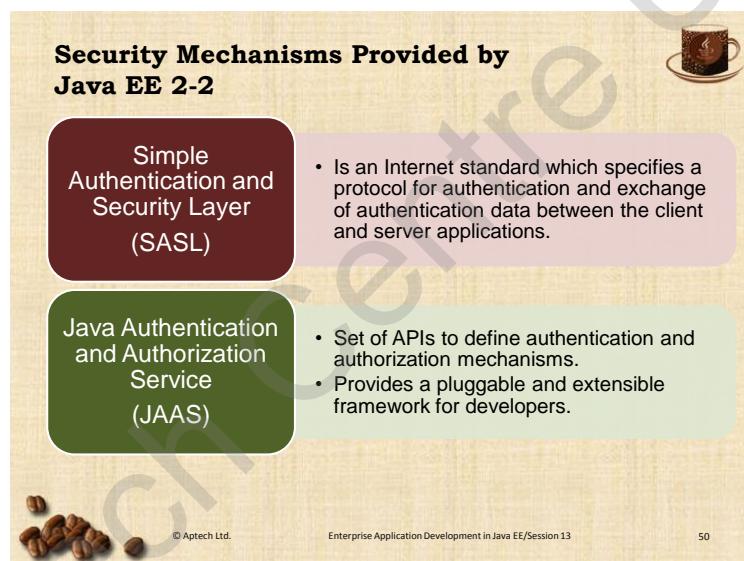
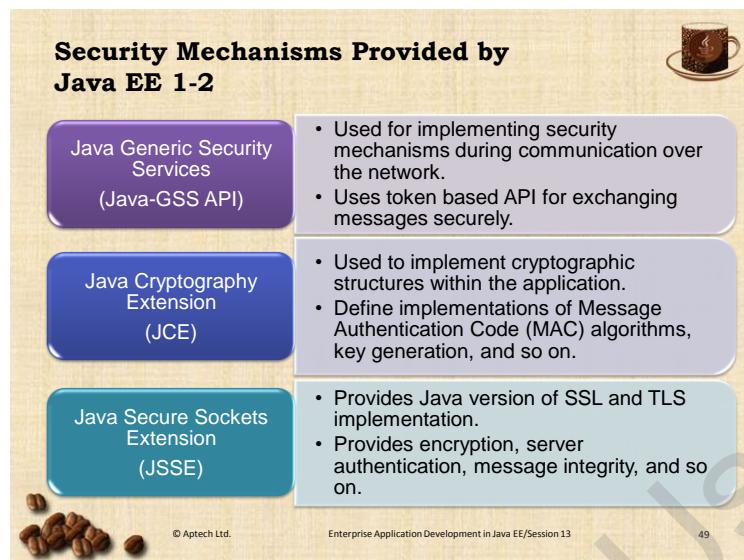
Which interface do we can use to provide programmatic security to the enterprise bean?

Answer:

The `EJBContext` interface provides various methods to implement security among the enterprise beans.

Slides 49 and 50

Let us understand the different APIs which can be used to implement security mechanisms.



Use slides 49 and 50 to explain the APIs available in Java EE 7 to implement security mechanisms in the application.

- Java Generic Security Services (Java GSS-API)** - It is used for implementing security mechanisms during communication over the network. This is a token based API used for exchanging messages securely.
- Java Cryptography Extension (JCE)** – JCE is used to implement cryptographic structures within the application. Developers can define implementations of Message Authentication Code (MAC) algorithms, key generation, and so on. The API also enables generating cipher text in the application. It allows incorporation of cryptographic structures in the application at various levels.
- Java Secure Sockets Extension (JSSE)** – It provides a Java version of implementation of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It provides the functionality of encryption, server authentication, message integrity, and so on.

- **Simple Authentication and Security Layer (SASL)** - It is an Internet standard which specifies a protocol for authentication and exchange of authentication data between the client and server applications.
- **Java Authentication and Authorization Service (JAAS)** – It is a set of APIs used to define authentication and authorization mechanisms of the application. It provides a pluggable and extensible framework for developers.

In-Class Question:

After you finish explaining the different Java EE APIs, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which API would you use if you have to implement cryptographic functions for security in an application?

Answer:

Java Cryptography Extension.

Slide 51

Let us understand Java Authentication and Authorization Service (JAAS).

Java Authentication and Authorization Service (JAAS)

- ❑ JAAS implements Pluggable Authentication Model (PAM) framework.
- ❑ JAAS provides the following classes and interfaces to be used by developers for implementing security mechanisms.
- ❑ Following are the components of the core class library:
 - LoginModule
 - LoginContext
 - Subject
 - Principal
- ❑ Other classes include:
 - CallbackHandler
 - Credentials

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 51

Use slide 51 to explain JAAS. JAAS provides authentication and authorization mechanisms for enterprise applications. These classes can be used by developers to implement security mechanisms.

There are four core classes in the application:

- **LoginModule** – LoginModule is an interface which is implemented by the application to define the login process. Application developer writes the code for the authentication and authorization process.
- **LoginContext** – LoginContext initiates the authentication process. It creates a Subject in the application execution environment.
- **Subject** – An instance of Subject is used to represent the user who is trying to access the protected application resources.
- **Principal** – Principal refers to the properties associated with the Subject which are used in the authentication process.

Apart from these core classes JAAS also has CallbackHandler and Credentials classes.

Tips:

An application implements a CallbackHandler and passes it to underlying security services so that they may interact with the application to retrieve specific authentication data, such as usernames and passwords, or to display certain information, such as error and warning messages.

Credentials class is used to handle the username, password, and other credentials of the application.

Principals

J2EE security architecture does not deal with user identities directly. It deals with abstract user identities called principal. A principal is an entity that can be authenticated by the system. This is typically an end user or any other service requesting access to the application. The principal is identified by a name; often the username is also the principal for that end user.

Slide 52

Let us understand the steps involved in JAAS authentication process.

JAAS Authentication 1-2

□ JAAS authentication process involves the following steps:

1. Create a `LoginContext`, the client application accesses the authentication mechanism through an instance of `LoginContext`.
2. The `LoginContext` module accesses the `LoginModule`, which is defined in the configuration file.
3. The authentication is performed through the `LoginModule`.
4. In the authentication process, a `CallBackHandler` is used to communicate with the client and acquire authentication information such as username, password, and so on.
5. If the authentication process fails or login process was unsuccessful, a `LoginException` is thrown.
6. `LoginContext` is used to logout from the session.

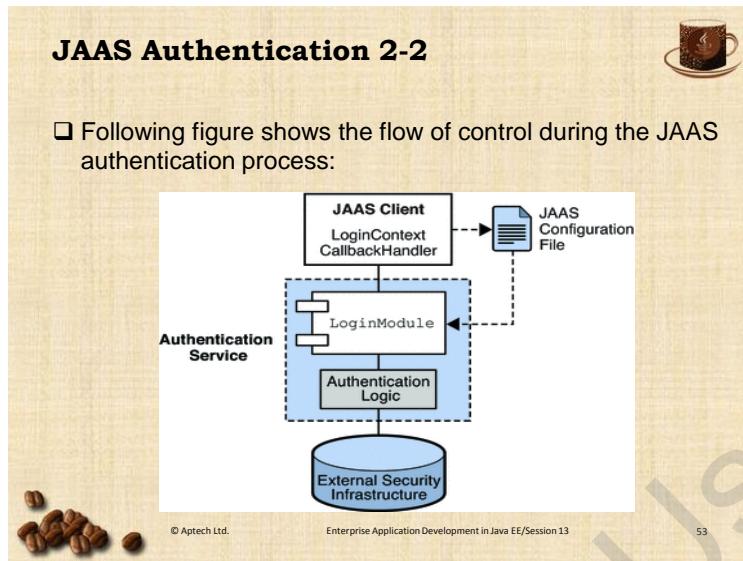
© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 52

Use slide 52 to explain the steps involved in JAAS authentication process.

1. Create a `LoginContext`, the client application accesses the authentication mechanism through an instance of `LoginContext`.
2. The `LoginContext` module accesses the `LoginModule`, which is defined in the configuration file.
3. The authentication is performed through the `LoginModule`.
4. In the authentication process, a `CallBackHandler` is used to communicate with the client and acquire authentication information such as username, password, and so on.
5. If the authentication process fails or login process was unsuccessful, a `LoginException` is thrown.
6. `LoginContext` is used to logout from the session.

Slide 53

Let us understand the JAAS authentication process graphically.

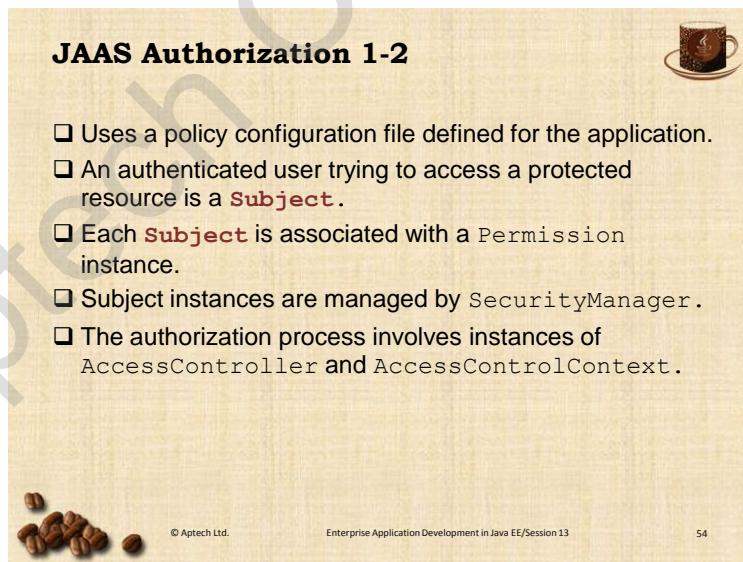


Use slide 53 to explain the JAAS authentication process graphically. The authentication process is initiated by the JAAS client. The client initiates the process from a `LoginContext`.

The `LoginModule` on the authentication service is invoked which accepts the credentials from the client and processes them.

Slides 54 and 55

Let us understand the JAAS authorization process.



JAAS Authorization 2-2

Following are the steps involved in the authorization process:

1. The `doAs()` method of `Subject` class is invoked to associate a role to the authenticated user.
2. The `SecurityManager` checks the permissions associated with the `Subject` using `checkPermission()` method. It in turn invokes the `AccessController`.
3. `AccessController` performs the required check and updates the `AccessControlContext` with the `Subject` and its associated permissions.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 55

Use slides 54 and 55 to explain the JAAS authorization process.

Authorization process is dependent on the security policy defined by the application. It uses a policy configuration file defined for the application. The authenticated user who is trying to access a protected resource is an instance of `Subject`. Each `Subject` is associated with a permission instance indicating the access rights of the user. The subjects are managed by a `SecurityManager` instance in the application.

The authorization process also involves instances of `AccessController` and `AccessControlContext` through which the permissions of the authenticated user are checked and granted access.

The process of authorization is carried out in the following steps:

1. The `doAs()` method of `Subject` class is invoked to associate a role to the authenticated user.
2. The `SecurityManager` checks the permissions associated with the `Subject` using `checkPermission()` method. It in turn invokes the `AccessController`.
3. `AccessController` performs the required check and updates the `AccessControlContext` with the `Subject` and its associated permissions.

Additional References:

To prepare an example on JAAS for the demonstration, refer the following link:

<http://www.greenkode.com/2011/09/user-authentication-and-authorization-using-jas-and-servlet-3-0-login/>.

Slide 56

Let us understand the concept of propagating the security identity among the application components.

Propagating a Security Identity

- Following are different options for propagating the user identity:
 - The identity of the entity through which the user accessed the first entity can be propagated to the second entity by default.
 - By configuring a 'Run-as' identity for the bean.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 56

Use slide 56 to explain the process of propagating the security identity.

If a client is authenticated by one of the application components, then that authenticated identity should be propagated to all the other application components.

Access to other components should not prompt for credentials again and again.

The user identity of the authenticated client should be propagated through one of the following options:

- The identity of the entity through which the user accessed the first entity can be propagated to the second entity by default. This technique is used when the intermediate entity is a trusted entity.
- When the target enterprise bean expects a specific identity, then the expected identity is forwarded to the target enterprise bean. In order to propagate an identity to the target enterprise bean, a run-as identity is configured for the bean. The 'run-as' identity is one which is used by the enterprise bean when it makes calls. The 'run-as' identity is bound to a user whose role is determined after authentication. The `RunAs` annotation can be used to configure the 'run-as' identity or the propagated identity of an entity.

Tips:

EJB-tier authorization is carried out on the basis of the current security context. Security context is cached data that contains security details, such as username, password, and authorization about a principal.

In the EJB call chain, the default security context propagation is called propagated which enables the propagation of the default or the actual security context of the client. You can specify the propagated security context using the `<use-caller-identity>` element in the `ejb-jar.xml` deployment descriptor.

The following code snippet makes use of the `<use-caller-identity>` element. This element instructs the security subsystem to use the default security context.

```
<enterprise-beans>
<session>
...
<ejb-name>EmployeeBean</ejb-name>
...
<security-identity>
</use-caller-identity>
</security-identity>
</session>
</enterprise-beans>
```

Declarative Security Context

Declarative propagation is used if you want to hide or do not want to use the security context of the client. An additional `<run-as>` element is used in the `ejb-jar.xml` when using declarative propagation. This element contains a real security role name declared using the `<security-role>` element. When this EJB is called, the current security context will refer to this security role rather than the security role of the client.

The following code snippet declares makes use of `<run-as>` element. This element instructs the security subsystem to use the security context of the specified role name.

```
<enterprise-beans>
<session>
...
<ejb-name>EmployeeBean</ejb-name>
...
<security-identity>
  <run-as>
    <role-name>SuperAdmin</role-name>
  </run-as>
</security-identity>
</session>
</enterprise-beans>
```

Additional References:

To get more information on configuring propagated security identity, refer the following link:
<http://docs.oracle.com/cd/E19798-01/821-1841/bnbyr/index.html>.

Slide 57

Let us understand the security of clients in the application.

Securing Application Clients

- The security requirements of application clients are similar to that of EJB components.
- The application client authenticates the users accessing it either:
 - When the application client starts.
 - When the user is trying to access a protected resource in the application.
- The application client can use a `LoginModule` object to gather the user information.
- The `CallBackHandler` instances can further carry out the authentication process.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 57

Use slide 57 to explain the implementation of security mechanisms for application client.

Provide a brief explanation of securing application clients.

The mechanisms used for EJB components can be used for application clients as well. Clients can make use of the services provided by the underlying platform.

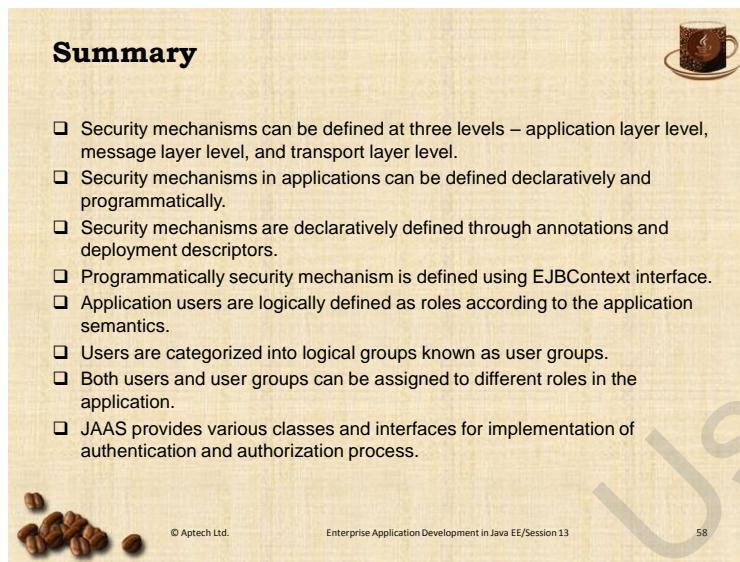
An application client makes use of an authentication service provided by the application client container for authenticating its users. The container can authenticate the user either when the application is started or when a protected resource is accessed.

An application client can provide a class, called a login module, to gather authentication data. The `javax.security.auth.callback.CallbackHandler` interface must be implemented, and the class name must be specified in its deployment descriptor. The application's callback handler must fully support Callback objects specified in the `javax.security.auth.callback` package.

Here, you can assign the students to work further on how to secure application clients as part of the implementation which needs to be discussed in the lab session.

Slide 58

Let us summarize the session.



Summary

- ❑ Security mechanisms can be defined at three levels – application layer level, message layer level, and transport layer level.
- ❑ Security mechanisms in applications can be defined declaratively and programmatically.
- ❑ Security mechanisms are declaratively defined through annotations and deployment descriptors.
- ❑ Programmatically security mechanism is defined using EJBContext interface.
- ❑ Application users are logically defined as roles according to the application semantics.
- ❑ Users are categorized into logical groups known as user groups.
- ❑ Both users and user groups can be assigned to different roles in the application.
- ❑ JAAS provides various classes and interfaces for implementation of authentication and authorization process.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 13 58

Using slide 58, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

13.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the EJB Timer Service that is offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 14 – EJB Timer Service

14.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

14.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe how to use a timer in an enterprise application
- Explain the different types of timers in EJB
- Describe how to handle timers in an application
- Describe annotations associated with timers
- Describe operations performed on `Timer` objects

14.1.2 Teaching Skills

To teach this session successfully, you should be aware of EJB timer service provided by Java EE and how developers can schedule operations in the application with the help of the timer service. You should be aware of different types of timers that can be defined through EJB timer service.

You should also be aware of the timer handling mechanisms and annotations that are used to define timers in the application.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

Objectives

- ❑ Describe how to use a timer in an enterprise application
- ❑ Explain the different types of timers in EJB
- ❑ Describe how to handle timers in an application
- ❑ Describe annotations associated with timers
- ❑ Describe operations performed on Timer objects

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 2

Tell them that they will be introduced to EJB timer service. Java provides timer service to enable application developers to define various application operations based on a timer without any manual intervention. The timer can be set according to the absolute clock time or according to relative clock.

The session also explains about different types of timers that are provided by Java EE. It also provides different annotations through which timers can be defined for applications. Developers can define various operations to be performed automatically by defining Timer objects.

14.2 In-Class Explanations

Slide 3

Let us understand the timer service.

Introduction

- ❑ Enterprise applications require to schedule various tasks such as auditing, generating reports, and so on.
- ❑ These tasks are scheduled to be executed at regular intervals or at a prescribed time.
- ❑ Java EE provides EJB timers to schedule tasks in the application code.
- ❑ EJB TimerService was first introduced in EJB 2.1.
- ❑ The TimerService allows the EJB container to register certain EJB methods to be invoked at a scheduled time.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 3

Use slide 3 to explain the timer service.

Applications have different tasks which are to be executed at specific times or repeated at regular intervals of time. Java EE provides EJB timer service to define these tasks which are to be executed at a specific time.

The timer service allows the EJB container to register certain EJB methods to be invoked at a scheduled time. The timer service is used to implement long running processes therefore, the timer is expected to survive server crashes and shutdowns.

Tips:

EJB 2.1 introduced the support for Web services and timer service. The Web service allowed a Stateless Session bean to expose an endpoint interface and timer service allowed EJBs to be invoked at scheduled intervals. Then, EJB 3.1 has enhanced timer support. Finally, EJB 3.2 enhanced EJB timer service features and added it as a part of EJBLite subset.

Slides 4 and 5

Let us understand the services provided by the TimerService.

Timer Service 1-2



- ❑ The EJB Timer service is a container-managed service.
- ❑ Timer objects are created by interacting with the TimerService of the container.
- ❑ Following are the functions which can be implemented by the developer:
 - Create a timer callback function**
 - Timer object is created with either a calendar time or number of seconds. When the calendar time is reached or when the timer expires, the operation to be performed is defined in the timer callback function.
 - Process a timer callback function**
 - Callback method has to be invoked and executed once the timer expires or the scheduled time is reached.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 4

Timer Service 2-2



- Check and interact with the timer callback notification object**
 - When an object generates a timer callback notification, the TimerService enables interacting with the notification generating object.
- Obtain a list of outstanding timer notifications**
 - When there are multiple timer callback notifications, the TimerService enables listing outstanding timer notifications.
- Cancelling a timer notification**
 - The TimerService enables the application to cancel the timer notification according to the application requirement.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 5

Use slides 4 and 5 to explain the services provided by the TimerService.

The EJB Timer Service is a container-managed service. The developer can create and implement Timers through Timer objects. They are created by interacting with TimerService of the container and through annotations.

TimerService can instantiate Timer objects which can schedule various application related tasks in the application.

Following are the services provided by the TimerService:

1. **Create a timer callback function** - When a developer creates a Timer object, it is created with either a prescribed calendar time or number of seconds in which the timer can expire.

When the calendar time is reached or when the timer expires, the operation to be performed is defined in the timer callback function. The callback function enables the developer to define the operations to be performed when the timer expires.

2. **Process a timer callback function** – The timer callback function has to be invoked and executed once the timer expires or the scheduled time is reached.
3. **Check and interact with the timer callback notification object** - When an object generates a timer callback notification, the `TimerService` enables interacting with the notification generating object.
4. **Obtain a list of outstanding timer notifications** – When there are multiple timer callback notifications, the `TimerService` enables listing outstanding timer notifications.
5. **Cancelling a timer notification** – The `TimerService` enables the application to cancel the timer notification according to the application requirement.

Tips:

You should use timer service for application level functionalities, but not for real time events. Some of the examples where timer services can be useful are as follows:

- In a reporting application, a timer can be created to print newly filed expenses at regular intervals.
- A timer in a bug tracking application emails a list of open bugs to team members at scheduled time.
- A timer in a human resources application emails a list of public holidays to all the employees on the 1st of January every year.

Also, note that the timer service cannot be created for Stateful Session bean.

In-Class Question:

After you finish explaining services provided by the `TimerService`, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which of the following services has to be invoked and executed once the timer expires or the scheduled time is reached?

Answer:

Process a timer callback function.

Slide 6

Let us understand the timer notification.

Timer Notification

- ❑ Timer notification can be defined at the timer expiry event.
- ❑ The notification is handled through timer callback methods.
- ❑ Developer can use any of the following methods to designate a method as timer notification callback method:
 - Annotate the method with `@Timeout` annotation
 - Implement the `javax.ejb.TimedObject` interface and define `ejbTimeOut()` method
 - Annotate with `@Schedule` annotation

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 6

Use slide 6 to explain the concept of timer notification.

Timer notification can be defined as the event of timer expiry. When a timer defined in the application expires the container of the application has to be notified. The container will then, in turn invoke the callback method defined for this timer notification.

The timer callback methods are to be annotated with `@TimeOut` or through `@Schedule` annotation.

An alternative way of executing the timeout event is implementing the `ejbTimeOut()` method of `javax.ejb.TimedObject` interface.

Additional References:

To create a timer service for a Stateless Session bean, refer to this link:

<http://examples.javacodegeeks.com/enterprise-java/ejb3/timer/ejb-timer-service-example/>.

In-Class Question:

After you finish explaining timer notification, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



When does the timer notification can be defined?

Answer:

Timer notification can be defined at the timer expiry event.

Slides 7 to 10

Let us understand how to create a Timer callback notification.

Creating a Timer Callback Notification 1-4



Following code snippet demonstrates creating and using a Timer callback notification:

```
import java.util.Timer;
import java.util.TimerTask;

public class TimerDemo {
    Timer timer;

    public TimerDemo(int seconds) {
        timer = new Timer();
        timer.schedule(new Reminder(), seconds * 1000);
    }
}
```

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 14 7

Creating a Timer Callback Notification 2-4



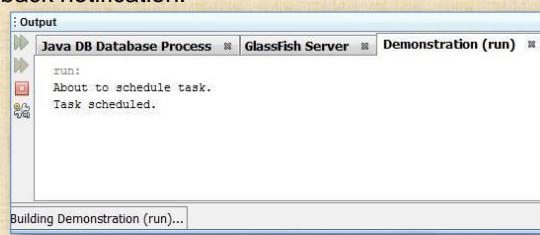
```
class Reminder extends TimerTask {
    public void run() {
        System.out.println("You have a meeting!");
        System.exit(0);
    }
    public static void main(String args[]) {
        System.out.println("About to schedule a reminder.");
        TimerDemo T = new TimerDemo(10);
        System.out.println("Reminder scheduled after 10
seconds");
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 8

Creating a Timer Callback Notification 3-4



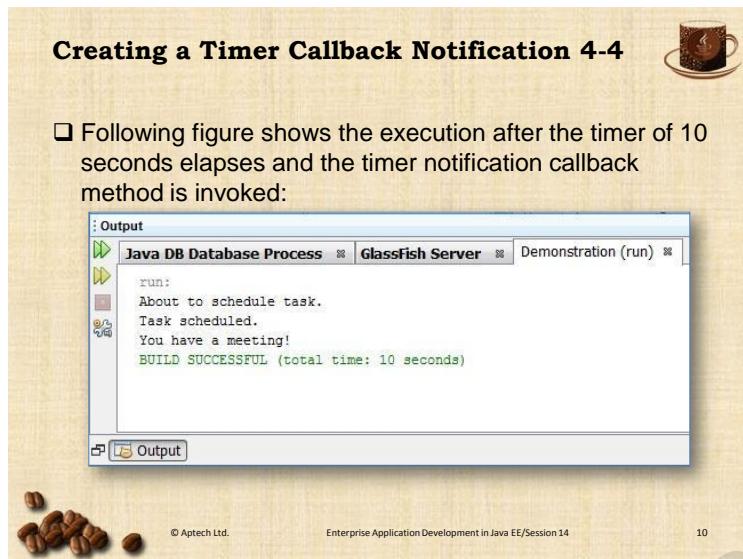
Following figure shows the output before the timer callback notification:



The screenshot shows the Java code for the Reminder class and its execution output. The output window displays:

```
Java DB Database Process > GlassFish Server > Demonstration (run) >
run:
About to schedule task.
Task scheduled.
```

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 14 9



Use slides 7 to 10 to explain the process of creating a Timer callback notification.

Java EE provides various classes such as `TimerTask`, `TimerHandler`, `TimerService`, `TimedObject`, and so on to handle the timers in the application.

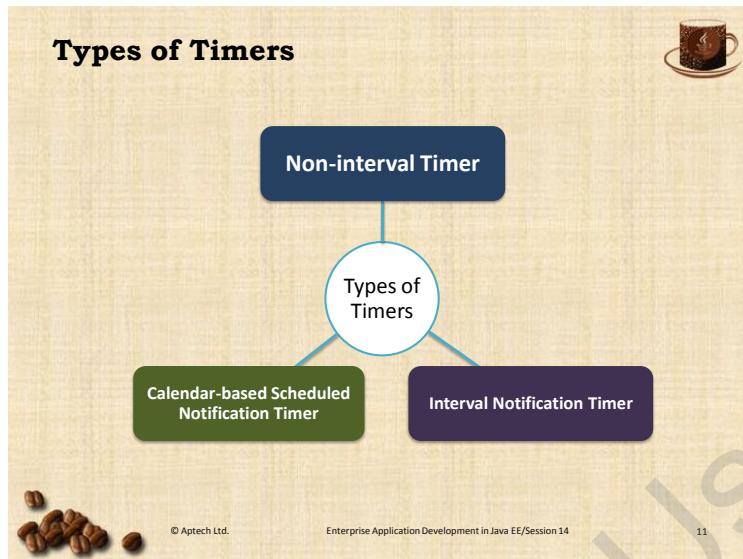
A `TimerDemo` class is created in which the `Timer` object is instantiated. The `TimerDemo` has a parameterized constructor which accepts the number of seconds after which the timer can expire. Such a timer is known as a relative timer where the timer is defined to expire after certain time period.

The application also has a `Reminder` class which extends the `TimerTask` class. The `Reminder` class defines the `run()` method of the class, which is executed by the container after receiving a timer notification. The `main()` method schedules the timer for the application and executes it.

Use slides 9 and 10 to explain the output of the defined Timer class. Slide 10 shows the execution after the timer of 10 seconds elapses and the timer notification callback method is invoked.

Slide 11

Let us understand different types of timers.



Use slide 11 to explain different types of timers.

EJB timer service offers three types of timers:

- **Non-interval timers** – These timers are used to create a notification based on absolute time or the notification relative to the execution time, like the one demonstrated in the code earlier.
- **Calendar-based scheduled notification timer** - These timers are created based on calendar expressions.
- **Interval notification timer** – These timers are used to generate recurring timer notifications at regular intervals.

Slide 12

Let us understand non-interval notification timers.

Non-Interval Notification Timers

- ❑ Non-interval notification timers create a notification based on the absolute time or delay interval.
- ❑ Notifications are generated based on a `Date` object or delay interval specified in terms of seconds.
- ❑ They are non repetitive.
- ❑ Two variants of non-interval notification timers:
 - Absolute Time Single Event Timer
 - Relative Time Single Event Timer

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 12

Use slide 12 to explain the non-interval notification timers.

Developers can use these `Timer` objects to generate according to either absolute time represented through a `Date` object or by specifying the delay interval through the number of milliseconds which can elapse.

These timers cannot be repeated. There are two variants to non-interval notification timers.

They are:

- Absolute Time Single Event Timer
- Relative Time Single Event Timer

Slide 13

Let us understand the absolute time single event timer.

Absolute Time Single Event Timer

- ❑ Invokes a timer callback method only once at a specific time in the future.
- ❑ The timer is created using `createTimer()` method.
- ❑ Following is the syntax for `createTimer()` method:

Syntax:

```
Timer createTimer(Date expiration, Serializable info)
```

- The method returns a `Timer` object, which expires at the date specified as the `Date` parameter.
- The `info` parameter represents the application information to be delivered along with the notification.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 13

Use slide 13 to explain absolute time single event timer.

Absolute time single event timer invokes a timer callback method only once and at a specific time in the future.

Following is the prototype of the `createTimer()` method used for creating an absolute time single event timer:

Syntax:

```
Timer createTimer(Date expiration, Serializable info)
```

The method returns a `Timer` object, which expires at the date specified as the `Date` parameter. The `info` parameter represents the application information to be delivered along with the notification.

Slide 14

Let us understand the Relative Time Single Event Timer.

Relative Time Single Event Timer

- ❑ Defines a timer callback which is invoked after a specific time in the future.
- ❑ The timer is created using `createTimer()` method.
- ❑ Following is the syntax for `createTimer()` method used to create a relative timer:

Syntax:

```
Timer createTimer(long duration, Serializable info)
```

- ❑ The duration parameter is a milliseconds unit.
- ❑ Therefore, an appropriate long value has to be specified as the parameter.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 14

Use slide 14 to explain Relative Time Single Event Timer.

Relative Time Single Event Timer is used to define a timer callback method which is invoked after a specific time in the future. Following is the prototype of the `createTimer()` method used for creating a timer which is invoked relative to a specific time:

Syntax:

```
Timer createTimer(long duration, Serializable info)
```

The duration parameter is a milliseconds unit. Therefore, an appropriate long value has to be specified as the parameter.

Slide 15

Let us understand the interval notification timers.

Interval Notification Timers 1-2

- ❑ The timer notifications are generated at regular intervals.
- ❑ Used when certain task has to be executed at regular intervals of time.
- ❑ There are two variants of interval notification timers:
 - Interval timer with initial absolute timer
 - Interval timer with initial relative timer

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 15

Use slide 15 to explain interval notification timers.

These timers are used when certain operations are to be invoked at regular intervals of time.

There are two variants of the interval notification timers - Interval timer with initial absolute timer and Interval timer with initial relative timer.

Slide 16

Let us understand the variants of interval notification timers.

Interval Notification Timers 2-2

Interval timer with initial absolute timer

- First timer notification specified by an absolute value
- Duration of consecutive intervals are specified
- **Syntax:**
`createTimer (Date initialExpiration, long interval, Serializable info)`

Interval timer with relative initial timer

- First timer notification by a relative value, which is relative to the time when the timer is scheduled.
- Consecutive timer notifications are specified through the long value.
- **Syntax:**
`createTimer(long initialDuration, long interval, Serializable info)`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 16

Use slide 16 to explain the variants of interval notification timers.

Interval timer with initial absolute timer

It has the first timer notification specified by an absolute value through a Date parameter and the duration of consecutive intervals are specified.

Following is the prototype of the `createTimer()` method used to create an interval timer with initial absolute timer:

Syntax:

```
createTimer (Date initialExpiration, long interval, Serializable info)
```

Interval timer with initial relative timer

It has the first timer notification specified by a relative value, which is relative to the time when the timer is scheduled. The consecutive timer notifications are specified through the long value.

Following is the prototype of the `createTimer()` method used to create an interval timer with initial relative timer:

Syntax:

```
createTimer(long initialDuration, long interval, Serializable info)
```

Slide 17

Let us understand calendar-based notification timers.

Calendar-Based Notification Timers 1-3

- ❑ Calendar-based notification timers generate notifications based on a schedule created on the basis of absolute calendar dates.
- ❑ Calendar-based expressions can be used along with @Schedule annotation.
- ❑ Schedule class is used to define the schedule of timer notifications in the application.
- ❑ Following is a sample calendar expression:
 - year = "2008,2012,2016" dayOfWeek = "Sat,Sun" hour = "10" minute = "0-10,30,40"

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 17

Use slide 17 to explain calendar-based notification timers. The timers are scheduled based on calendar expressions. The calendar expressions can be defined in different formats.

Calendar-based notification timers generate notifications based on a schedule created on the basis of absolute calendar dates. Developers can define the schedule through calendar-based expressions.

Slide 18

Let us understand the symbols used to represent calendar-based expressions.

Calendar-Based Notification Timers 2-3

□ Following table shows the attributes that can be used in calendar expressions:

Attribute	Default Value	Possible Values
second	0	[0,59]
minute	0	[0,59]
hour	0	[0,23]
dayofWeek	*	[0-7] or { "Sun","Mon","Tue","Wed","Thu","Fri","Sat"}. Both 0 and 7 refer to Sunday
month		[1-12],{"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"}
dayofMonth	*	[1-31] or {"1st","2nd","3rd"... and so on} or "Last" which implies the last day of the month.
year	*	Four digit value
timezone		Timezone according to tz database

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 18

Use slide 18 to explain different attributes of calendar-based expressions.

Following table gives the semantics of each of calendar expression:

Attribute	Default Value	Possible Values
second	0	[0,59]
minute	0	[0,59]
hour	0	[0,23]
dayofWeek	*	[0-7] or { "Sun","Mon","Tue","We d","Thu","Fri","Sat"}. Both 0 and 7 refer to Sunday
month		[1-12],{"Jan","Feb","Mar","A pr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"}
dayofMonth	*	[1-31] or {"1st","2nd","3rd"... and so on} or "Last" which implies the last day of the month
year	*	Four digit value
timezone		Timezone according to tz database

Slide 19

Let us understand different forms of calendar expressions.

Calendar-Based Notification Timers 3-3

- Following are the different forms in which a Calendar expression can accept data:
 - As a single value
• year = "2015" month = "May"
 - As a wild card expression
• month = "*" (highlighted)
 - As a list
• month = "May,Apr,Dec,Nov"
 - As a range expression
• dayofMonth = [10-20]
 - As increment expression
• second = "*/15"

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 19

Use slide 19 to demonstrate different forms of calendar expressions.

As a single value, the calendar expression can be specified as:

```
year = "2015" month = "May"
```

A wildcard expression can be used to specify a set of values in the calendar expression. Here is an instance of using a wild card:

```
month = "*" (highlighted)
```

This expression implies any of the possible values of the attribute.

List is a comma separated set of values which can be assigned to the attributes of the application. It can be used as follows:

```
month = "May,Apr,Dec,Nov" (highlighted)
```

The list specifies that the month attribute can have any one of the four values given in the list.

Range expression specifies a set of continuous values between two values. A range expression can be specified as follows:

```
dayofMonth = [10-20]
```

Increments expression is used in specifying interval timers where the starting point of the timer and their intervals are specified by separating it with a slash as follows:

```
second = "*/15" (highlighted)
```

This expression implies that the timer may be initiated at any instance, after initiating the timer it has to be incremented by 15 seconds.

Slide 20

Let us understand the code used to create a `Timer` object.

Creating a Timer Object 1-2

- ❑ To create a `Timer` object, the enterprise bean has to create an object reference of `TimerService`.
- ❑ Following code snippet demonstrates creating the reference using `getTimerService()` method:

```
private SessionContext SC ;
.....
TimerService TS = SC.getTimerService();
timerService.createTimer(1000*60, text);
...
```

- ❑ `Timer` objects are also created declaratively with `@Schedule` annotation.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 20

Use slide 20 to explain the code to create a `Timer` object.

The `TimerService` enables creation of the `Timer` object. A reference to the `TimerService` has to be created.

The reference to the `TimerService()` is obtained through `getTimerService()` method.

`createTimer()` method or `@Schedule` annotation can then be used to create the `Timer` object.

In-Class Question:

After you finish explaining the code, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



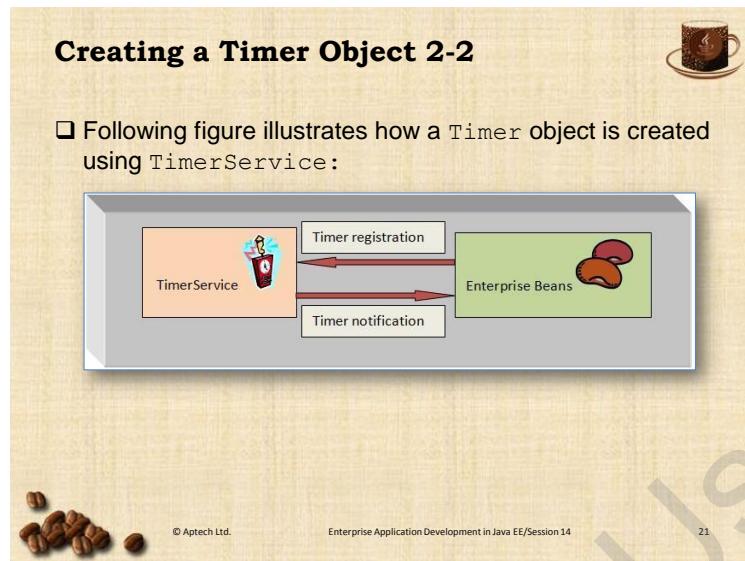
To audit the application log once in every three hours, which type of timer should be used?

Answer:

Interval notification timers, as they enable defining recurring timers.

Slide 21

Let us graphically understand the process for creating the Timer object.



Use slide 21 to explain the process of creating the Timer object.

The EJB which requires a Timer object acquires the reference of a TimerService and then registers a Timer object with it. When the Timer object defined expires then, the TimerService sends a notification to the EJB.

Slide 22

Let us understand the processing of a timer callback notification.

Processing a Timer Callback Notification 1-2

- ❑ A timer callback notification can be processed in one of the following ways:
 - Through an enterprise bean which implements `TimedObject` interface
 - Through a method prefixed `@Timeout` annotation
- ❑ Through a method annotated with `@Schedule` used to process the timer callback notification.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 22

Use slide 22 to explain the processing of timer callback notification.

The notification can be processed either by implementing the `TimedObject` interface. Developers have to implement the method `ejbTimeOut()`.

The notification callback can be defined by creating a method annotated with `@TimeOut` annotation or `@Schedule` annotation.

Additional References:

You can get additional information about creating automatic and programmatic timer service on this link: <http://docs.oracle.com/javaee/6/tutorial/doc/bnboy.html>.

Slide 23

Let us understand the methods used for processing timer callback notifications.

Processing a Timer Callback Notification 2-2

Using the `TimedObject` interface

- The `TimedObject` interface has only one method `ejbTimeOut()` which accepts the `Timer` object which it has to handle as a parameter.

Using `@TimeOut` annotation

- A timeout callback method is defined in the bean class and annotated with `@Timeout` annotation.

Using `@Schedule` annotation

- The method annotated with `@Schedule` annotation receives notifications from the `TimerService`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 23

Use slide 23 to explain the methods used for timer callback notifications.

The `TimedObject` interface has only one method `ejbTimeOut()`. This method accepts the `Timer` object which it has to handle as a parameter. The prototype of the `ejbTimeOut()` method is as follows:

Syntax:

```
public void ejbTimeOut(Timer timer);
```

A timeout callback method in the bean class is annotated with `@Timeout` annotation. The method annotated with `@Timeout` annotation receives timer notifications.

This annotation allows the container to create timers. It is responsible for automatic timer creation according to the schedule specified as attribute to the annotation. The method annotated with `@Schedule` annotation receives notifications from the `TimerService`.

Slide 24

Let us understand some guidelines to keep in mind while creating timer handler method.

Guidelines for Coding Timer Handler Method

- ❑ The developer should consider the following factors while creating timer handler methods:
 - Identifying the timer which has sent a notification
 - Handling application shutdowns
 - Transaction handling for timer notification callback methods
 - Propagating the security context to the timer callback notification
 - Handling non-persistent timers

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 24

Use slide 24 to explain the guidelines for coding a timer handler method.

While creating the timer handler method, the developer should consider the following factors:

- **Identifying the timer which has sent a notification**

The `info` parameter of the `createTimer()` method can be used for the purpose. The timer handler method can identify the method with the help of the `info` parameter received along with the timer notification.

- **Handling application shutdowns**

A timer has to survive application shutdown. The `TimerService` holds the timer notifications of the timers that expired when the application was shutdown. These timers are delivered when the application restarts.

- **Transaction handling for timer notification callback methods**

The `ejbTimeout()` method has to execute in a transaction context, then the callback method should have code which can handle transaction rollback. In case of transaction rollback the `ejbTimeOut()` method is invoked again.

- **Propagating the security context to the timer callback notification**

The timeout notification callback methods do not have the security context of the application client. The bean provider uses the run-as deployment descriptor to specify the security identity of the client and execute the timeout callback methods.

- **Handling non-persistent timers**

Persistent timers are functional even on application shut down or server crash. Applications can also create non-persistent timers, these timers are not active when the application shuts down or when the application crashes. Non persistent timers can be created either programmatically or declaratively.

Slide 25

Let us understand the method for managing Timer objects.

Managing Timer Objects

- ❑ The Timer interface provides various methods to manage timer objects in the applications.
- ❑ Developers may have to perform the following tasks while managing timers:
 - Interrogate a timer callback notification
 - Obtain a list of outstanding timers
 - Cancel timer notification

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 25

Use slide 25 to explain the process of managing Timer objects.

Developers have to implement certain tasks to manage timers in the application.

- Interrogate a timer callback notification
- Obtain a list of outstanding timers
- Cancel timer notification

Slide 26

Let us understand the methods provided by the `Timer` class.

Interrogating a Timer Callback Notification

Following are the methods provided by the `Timer` class for interrogating the `Timer` object:

- `getHandle()`
- `getInfo()`
- `getNextTimeOut()`
- `getSchedule()`
- `getTimeRemaining()`

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 26

Use slide 26 to explain the methods provided by the `Timer` class.

- **`getHandle()`** – This method returns a `Serializable` handle to the `Timer` object. This handle can be used to reference the `Timer` object later in the application.
- **`getInfo()`** – It returns the info associated with the `Timer` object. The `info` field is defined when the `Timer` object is created and can contain any type of data which is `Serializable`.
- **`getNextTimeout()`** – This method returns the object of `Date` type which indicates the date when the next timer would expire.
- **`getSchedule()`** – This method returns an object of type `ScheduleExpression`, which signifies the schedule of the timer.
- **`getTimeRemaining()`** – This method returns the number of milliseconds remaining for the time to elapse before the bean receives a timer notification.

Slide 27

Let us understand how developers can obtain a list of timers.

Obtaining a List of Outstanding Timers

- ❑ `getTimers()` method of the `TimerService` class is used to retrieve the list of outstanding timers.
- ❑ The prototype of `getTimers()` method is:
`Collection<Timer> getTimers();`
- ❑ `cancel()` method of `Timer` class is used to cancel the timers. If there are no timers existing then the bean would encounter a `NoSuchObjectException`.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 27

Use slide 27 to explain the method of obtaining the list of outstanding timers in the application.

Outstanding timers imply those timers which have not yet generated a notification.

In order to obtain the list of outstanding timers, bean class can execute the `getTimers()` method of the `TimerService` class. The prototype of the `getTimers()` method is:

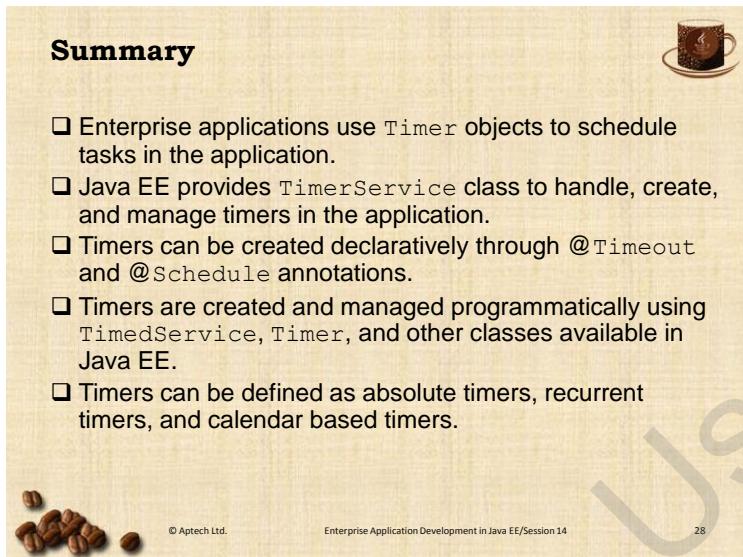
```
Collection<Timer> getTimers();
```

This method returns a collection of timers associated with the bean class.

To obtain a reference to the `TimerService` object, the enterprise bean can invoke a `getTimerService()` method on the associated `EJBContext`.

Slide 28

Let us summarize the session.



Summary

- ❑ Enterprise applications use Timer objects to schedule tasks in the application.
- ❑ Java EE provides TimerService class to handle, create, and manage timers in the application.
- ❑ Timers can be created declaratively through @Timeout and @Schedule annotations.
- ❑ Timers are created and managed programmatically using TimedService, Timer, and other classes available in Java EE.
- ❑ Timers can be defined as absolute timers, recurrent timers, and calendar based timers.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 14 28

Using slide 28, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

14.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Design patterns in Java EE platform that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 15 – EJB Design Patterns

15.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

15.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe design patterns
- Explain different types of design patterns
- Explain mostly commonly used Java EE design patterns
- Describe best practices in selecting EJB component in the enterprise applications

15.1.2 Teaching Skills

To teach this session successfully, you should be aware of what design patterns are and how you can use design patterns in the application development. You should prepare yourself with different types of design patterns provided by Java EE platform to create the enterprise applications in Java.

You should also be aware of a few examples of design patterns which can be used in Java applications.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students Slide 2 of the presentation.

Objectives

- Describe design patterns
- Explain different types of design patterns
- Explain mostly commonly used Java EE design patterns
- Describe best practices in selecting EJB component in the enterprise applications

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 2

Tell them that they will be introduced to design patterns, the purpose of design patterns, and how these patterns can be used in the application development process. They will understand the different categories of design patterns supported by Gang of Four (GOF) and various design patterns under each category.

Then, the session explains Java EE design patterns and also explains how they are used in designing enterprise solution. Finally, it lists the best practices followed when using EJB as business components in the enterprise applications.

15.2 In-Class Explanations

Slide 3

Let us understand the concept of design patterns.

Introduction 1-2

- ❑ In the world of programming, a previously used solution to a problem can be applied to a new problem by making relevant changes.
- ❑ This solution which has been used effectively in an earlier problem is described as a pattern.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 3

Use slide 3 to explain the concept of design patterns.

Discuss the solutions followed to solve a particular problem in everyday life. In day to day life, whenever a problem is encountered, most people tend to refer to a similar problem faced in the past. The problem had been successfully solved using some technique. That technique can be used as a model and can be tried out on the new problem by making relevant changes as per the current circumstances. Similarly, in the world of programming, a previously used solution to a problem can be applied to a new problem by making relevant changes. This solution which has been used effectively in an earlier problem is described as a pattern.

Design patterns aim at providing tried and tested solutions to application design problems. They identify a pattern from the problem based on the problem characteristics and parameters. This pattern is used to check if there is a similar problem solved earlier. If a similar problem has already been solved then, developers can use the solution of the solved problem to solve the current problem.

Slide 4

Let us define a pattern and understand how it is relevant to software pattern.

Introduction 2-2

- ❑ A pattern is a shape, model, or structure appearing again and again to create a design.
- ❑ Following figure shows a pattern of tiles arrangement in real world:

- The tiles are printed in such a way that they need to be arranged correctly, so that the print falls in the right place and a pattern is formed.
- Once a pattern is formed out of the combination of first set of tiles, it becomes a pattern or a solution for the rest of the floor.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 4

Use slide 4 to define what a pattern is and how it is relevant to application design.

A pattern in real world is a shape, model, or structure appearing again and again to create a design.

A design pattern is a proven solution to a problem which has been documented so that it can be used to easily handle similar problems in the future.

Explain the figure as shown on slide 4 that gives an example of printed tiles used for flooring. The tiles are printed in such a way that they need to be arranged correctly so that the print falls in the right place and a pattern is formed. Once a pattern is formed out of the combination of first set of tiles, the rest of the flooring is completed by copying the same pattern all over. The first set of tiles became a pattern or a solution for the rest of the floor.

Discuss with them various examples of application development where a particular pattern is followed to solve the problem. Any specific Java feature that has helped them in designing a pattern for their application.

Example of Java feature can be use of interfaces which allow the classes to provide specific implementation based on the requirements.

Then, tell them that design patterns used in software strategies are language-independent and are used for solving commonly encountered object-oriented design problems. Design patterns are proven techniques for implementing robust, reusable, and extensible object-oriented software.

Discuss with the students how design patterns are useful in object-oriented programming.

Slide 5

Let us understand the origin of design patterns.

Design Patterns

- The concept of design patterns was proposed by **Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides**, popularly known as **Gang of Four**.
- Design patterns
 - Are language independent strategies for solving object oriented design problems.
 - Are proven techniques for implementing robust, reusable, and extensible object-oriented software.
 - Helps in describing structures at a higher level of abstraction of classes, components, and instances.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 5

Use slide 5 to explain the origin of design patterns.

The concept of design patterns was proposed by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. They are popularly known as the '**Gang-of-Four**'. They describe design patterns as '**a solution to a problem in a context**'. There are three important aspects of a design pattern – problem, solution, and context.

Patterns have become an important concept in the development of object-oriented programming.

Design patterns speed up the process of finding a solution when another problem with similar characteristics is encountered. That is, it eliminates the need to 'reinvent the wheel'.

Design patterns provide reliable solutions to the problem, they considerably reduce the resources consumed while developing large enterprise applications.

In-Class Question:

After you finish explaining design patterns, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



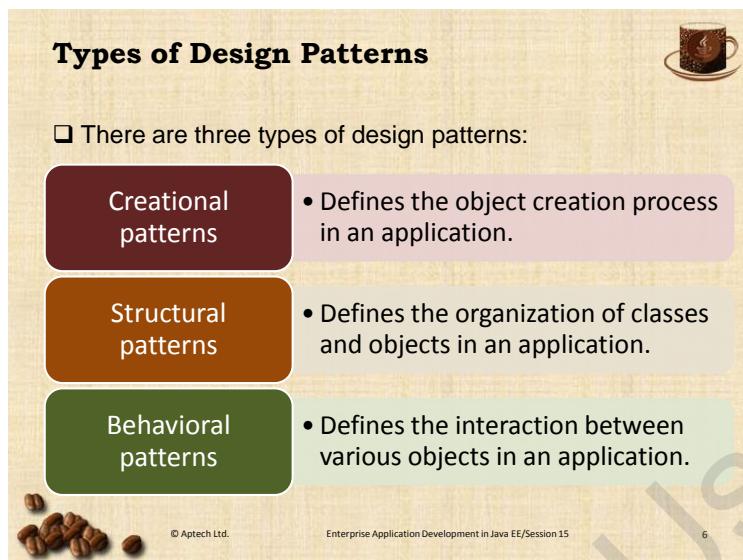
How Gang-of-Four describe the design patterns?

Answer:

They describe design patterns as '**a solution to a problem in a context**'.

Slide 6

Let us understand the types of design patterns.



Use slide 6 to explain different types of design patterns.

There are three types of design patterns:

Creational patterns – Define the object creation process in an application.

Structural patterns – Define the organization of classes and objects in an application.

Behavioral patterns – Define the interaction between various objects in an application.

Additional References:

To get more information on design patterns, refer the following link:

<http://www.blackwasp.co.uk/GofPatterns.aspx>.

Slides 7 and 8

Let us understand creational patterns.

Creational Patterns 1-2



- ❑ Used in scenarios where object composition is crucial to the application.
- ❑ Allow developers to configure a system with objects which vary widely in structure and functionality. Following table shows the list of creational patterns:

Pattern	Description
Singleton	Singleton pattern restricts the number of instances of a class to only one. It is used to implement tasks such as application logging and so on.
Factory	Factory pattern is used to create multiple instances of a class. This is used in a scenario where there is a super class with multiple subclasses. Based on the user input an instance of appropriate sub class is created.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 15 7

Creational Patterns 2-2



Pattern	Description
Abstract Factory	Abstract Factory pattern is similar to factory pattern. It is used to provide an interface for creating instances of sub classes in an inheritance structure without defining concrete classes.
Builder	Builder pattern is used when the instances of the class have large number of optional parameters.
Prototype	Prototype pattern is used when creating an instance of the class is expensive and resource intensive.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 15 8

Use slides 7 and 8 to explain different types of creational patterns.

Creational patterns are useful in scenarios where the object composition is crucial for the application. They enable creating objects, deciding the properties, and defining the methods of the objects based on the situation. They provide flexibility in terms of deciding what properties of the class are used in the application and what should be the behavior of the instantiated object.

Then, explain the table as shown on slides 7 and 8 to explain different creational patterns and their uses.

Additional References:

To know more about each of the creational pattern with example, you can follow the given links:

<http://www.blackwasp.co.uk/Builder.aspx>

<http://www.blackwasp.co.uk/AbstractFactory.aspx>

<http://www.blackwasp.co.uk/Prototype.aspx>

<http://www.blackwasp.co.uk/FactoryMethod.aspx>

<http://www.blackwasp.co.uk/Singleton.aspx>

Slides 9 to 11

Let us understand structural design patterns.

Structural Patterns 1-3



- ❑ Structural patterns define the structure of multiple classes and objects to function together.
- ❑ They define how classes and objects in the application combine to form a large structure.
- ❑ These patterns are used to realise the relationships among the classes of the application through object-oriented mechanisms such as abstraction and inheritance.
- ❑ Following table shows the list of structural patterns:

Pattern	Description
Adapter	Adapter pattern is used to bridge the structural gap between two interfaces.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 15

9

Structural Patterns 2-3



Pattern	Description
Composite	Composite patterns are used when composite objects should be treated like primitive objects.
Proxy	Proxy pattern is used when the application wants to provide a controlled access to certain functionality of the application. This pattern allows an object to provide a place holder for another object so that it can access it in a controlled manner.
Flyweight	Flyweight pattern is used when the application has to create large number of objects of a particular class. This is applied when the application runs on low memory devices as it allows sharing of memory.
Façade	Facade pattern is used for defining the interaction of client systems with the application.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 15

10

Structural Patterns 3-3



Pattern	Description
Bridge	Bridge pattern separates the interfaces from implementations. It is a structural pattern which aims at hiding the implementation details from the clients.
Decorator	Decorator design pattern enables modifying the functionality of an object at runtime. It also isolates other instances of the same class from being modified.



© Aptech Ltd.

Enterprise Application Development in Java EE/Session 15

11

Use slides 9 to 11 to explain structural design patterns.

Structural design patterns suggest solutions to problems where multiple classes of the application interact with each other. They define how objects and classes in the application combine together to form a large structure.

This pattern is essential for allowing independently developed classes and interfaces to work together. The focus of this pattern is how the classes are inherited from each other and how they are aggregated from other classes. It is used to realize the relationships among the classes of the application through object-oriented mechanisms such as abstraction and inheritance.

Use slides 10 and 11 to explain different types of structural patterns as presented on the slides.

Additional References:

To get more information on structural patterns, you can follow the given links:

<http://www.blackwasp.co.uk/Adapter.aspx>

<http://www.blackwasp.co.uk/Bridge.aspx>

<http://www.blackwasp.co.uk/Composite.aspx>

<http://www.blackwasp.co.uk/Decorator.aspx>

<http://www.blackwasp.co.uk/Facade.aspx>

<http://www.blackwasp.co.uk/Proxy.aspx>

In-Class Question:

After you finish explaining structural design patterns, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the main purpose of structural patterns?

Answer:

They define how objects and classes in the application combine together to form a large structure.

Slides 12 to 14

Let us understand behavioral design patterns.

Behavioral Patterns 1-3



- ❑ Behavioral patterns determine the interaction of the objects.
- ❑ They simplify the complex behavior by specifying the responsibilities of the objects and their communication method.
- ❑ Following table shows the list of behavioral patterns:

Pattern	Description
Template Method pattern	Template method pattern is a behavioral design pattern which defers the implementation of behavior of objects to sub classes in the hierarchy.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 15 12

Behavioral Patterns 2-3



Pattern	Description
Mediator pattern	Mediator pattern is used to establish a communication mechanism among different objects in the application. It implements loose coupling among the objects in the application.
Chain of responsibility pattern	Chain of responsibility pattern defines the method of handling the client request in the application. The client request is passed through a chain of objects while processing it.
Observer pattern	Observer pattern is used to track the state of an object in the application. The object which keeps track of the state is known as an Observer.
Strategy pattern	Strategy pattern enables the application to have multiple algorithms which can be used on certain event. The algorithm to be used is decided at runtime.
Command pattern	Command pattern is a behavioral pattern which implements loose coupling of the components. The application components execute in a request response model.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 15 13

Behavioral Patterns 3-3



Pattern	Description
State pattern	State pattern is used when the object changes its behavior based on its internal state.
Visitor pattern	Visitor pattern enables separating the operational logic to a separate class. It is used when an operation has to be performed on a group of similar objects.
Interpreter pattern	Interpreter pattern is used when the application has to analyze the syntax and semantics of input.
Iterator pattern	Iterator pattern is used when the application has to provide a standard way to traverse through a group of objects.
Memento pattern	Memento design pattern is used when the application has to save the state of the object and restore it for later use.

 © Aptech Ltd. Enterprise Application Development in Java EE/Session 15 14

Use slides 12 to 14 to explain behavioral design patterns.

Behavioral design patterns define the interaction of objects in the application. Behavioral patterns are those that determine the interaction of objects. These patterns are used when the behavior is complex. They simplify the complex behavior by specifying the responsibilities of the objects and their communication method. Most common behavioral pattern is observer pattern, which is used when an application is being implemented through a Model-View-Controller architecture. This separates the presentation of data from the actual data store.

Then, using slides 13 and 14, explain various behavioral patterns presented in the table.

Additional References:

To get the additional information on behavioral patterns, refer the given links:

<http://www.blackwasp.co.uk/ChainOfResponsibility.aspx>

<http://www.blackwasp.co.uk/Command.aspx>

<http://www.blackwasp.co.uk/Interpreter.aspx>

<http://www.blackwasp.co.uk/Mediator.aspx>

<http://www.blackwasp.co.uk/Iterator.aspx>

<http://www.blackwasp.co.uk/Observer.aspx>

<http://www.blackwasp.co.uk/State.aspx>

<http://www.blackwasp.co.uk/Strategy.aspx>

<http://www.blackwasp.co.uk/Visitor.aspx>

In-Class Question:

After you finish explaining behavioral design patterns, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



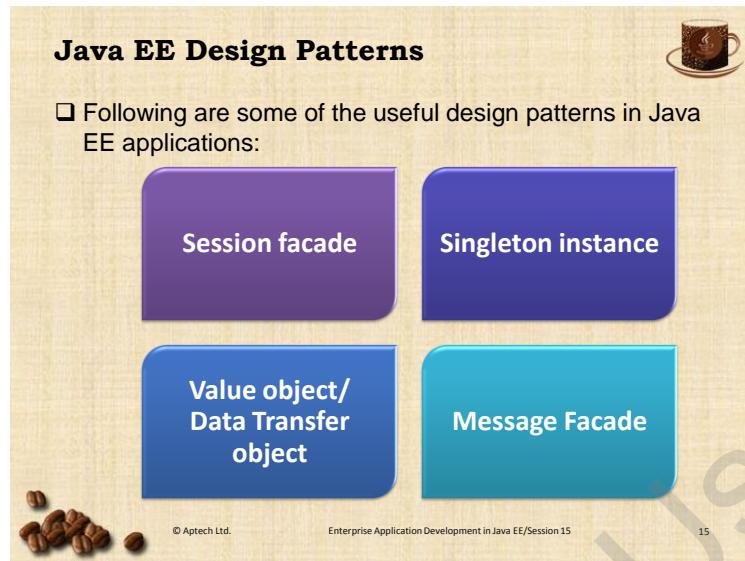
Which of the following is the most commonly used behavioral pattern?

Answer:

Observer.

Slide 15

Let us understand some patterns which are useful in creating Java enterprise applications.



Use slide 15 to explain different patterns which are useful in Java enterprise applications.

Tell them, apart from GOF design patterns, the SUN Microsystems identified a number of design pattern that simplifying the development of enterprise applications.

Some of the useful patterns commonly used in Java EE application development are as follows:

Session façade - The Session facade manages the interactions between the business data and business service objects that participate in the workflow of the application.

Singleton instance - A Singleton design pattern ensures that there is only one instance of a class in the application.

Value object/Data Transfer object - The value object pattern is used while implementing communication among different entities in the application.

Message Façade - Message façade pattern proposes asynchronous handling of method calls. The message façade pattern introduces an additional component such as a JMS server to asynchronously handle the messages among the application components.

Slide 16

Let us understand session façade pattern.

Session Façade 1-13

- ❑ Session façade abstracts the interaction of business objects by providing a service layer.
- ❑ The service layer exposes the required interfaces to the client.
- ❑ The session façade manages the interactions between the business data and business service objects in a workflow.
- ❑ Session façade enables decoupling of all the lower-level application components.
- ❑ Java EE applications implement session facade through session beans.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 16

Use slide 16 to explain session façade pattern. Session façade pattern defines the interactions of the business objects with the application clients.

In a complex enterprise application, following problems may arise:

- Tightly coupled application components, which lead to direct dependence between clients and business objects.
- Multiple method invocations between client and server, leading to network performance problems.
- Absence of uniform client access strategy, this may expose the business objects to malicious users.

Session façade addresses the mentioned problems. According to session façade pattern, the service layer exposes only the required interfaces to the client.

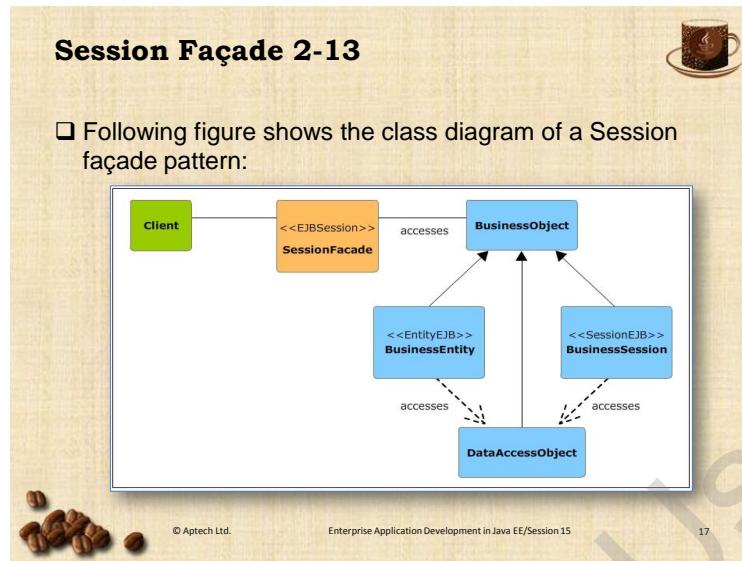
The Session Facade manages the interactions between the business data and business service objects that participate in the workflow of the application. This is achieved by using the session bean in the design pattern. The session bean provides a uniform interface to all the application clients by handling the relationship between the business objects. Session façade enables decoupling of all the lower-level application components.

The session bean manages the lifecycle of the business objects by creating, locating, modifying, and deleting them as per the requirements of the application.

Session façade is implemented by using the Session bean in the design pattern. The Session bean provides a uniform interface to all the application clients by handling the relationship between the business objects. Session façade enables decoupling of all the lower-level application components.

Slide 17

Let us graphically understand the implementation of a session facade using a class diagram.



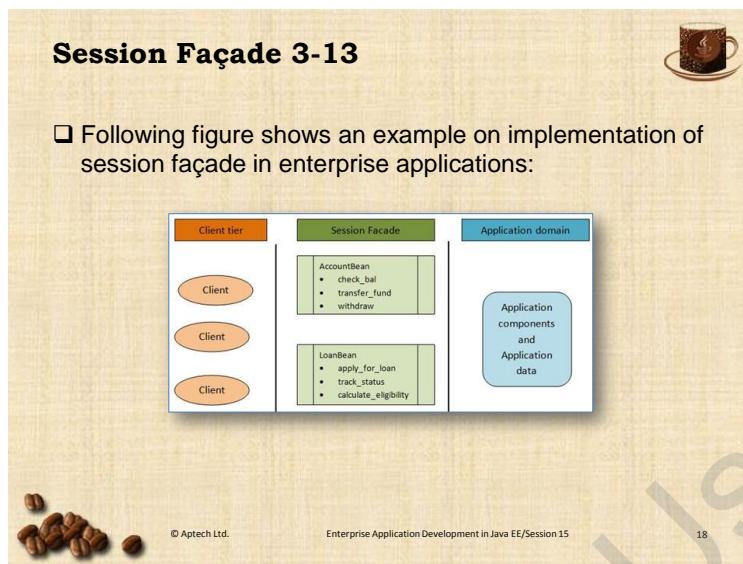
Use slide 17 to explain the class diagram of a session façade pattern.

Slide 17 demonstrates that the client of the session façade pattern accesses the business services through Session bean. The session façade manages the relationships between various business objects and provides a higher-level abstraction to the client. The business objects can be other Session beans, Entity beans, or Data Access Objects (DAO) that provide data or business services to the application.

In complex applications, there can be various session façade service objects that can sit between clients and business objects to provide interactions between them.

Slide 18

Let us understand session façade with the help of an example.



Use slide 18 to explain the implementation of session façade pattern in the context of a bank application.

Consider a scenario of bank application; the application has to implement the following tasks:

- Check balance
- Transfer money
- Withdraw money

The session façade pattern provides a common interface to implement all these functions in the application. Java application can define an enterprise bean such as Bank Teller bean which will implement these functionalities.

Similarly, for all the services pertaining to the Loan services offered by the Bank application, an enterprise bean can be created with the following methods:

- Apply for loan
- Track status
- Calculate loan eligibility

The session façade in this application is implemented through bean methods:

```
check_Balance(), transfer_fund(), withdraw(), apply_for_loan(),
track_status(), calculate_eligibility()
```

Slides 19 to 21

Let us understand the code for session façade implementation.

Session Façade 4-13



Following code snippet shows an entity class **Message** which is later accessed through session bean according to session façade pattern:

```
@Entity
public class Message implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String message;
    // Getter and Setter methods
    ...
    @Override
    public int hashCode() {
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 19

Session Façade 5-13



```
int hash = 0;
hash += (id != null ? id.hashCode() : 0);
return hash;
}
@Override
public boolean equals(Object object) {
if (!(object instanceof Message)) {
return false;
}
Message other = (Message) object;
if ((this.id == null && other.id != null) || (this.id != null && !this.
id.equals(other.id))) {
return false;
}
return true;
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 20

Session Façade 6-13



```
@Override
public String toString() {
    return "entities.Message[ id=" + id + " ]";
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 21

Use slides 19 to 21 to explain the implementation of an Entity class **Message**. The session façade is implemented for this Entity class.

The code snippet creates the attributes ‘`id`’ and ‘`message`’ respectively. The getter and setter methods are developed for the attributes. The `id` attribute is used as the primary key for accessing the entity in the database. There are other methods such as `equals()`, `toString()`, and `hashCode()` which are generated by API to manage the entity created by the entity class.

After creating the entity class, define the access to this entity through a session bean by creating a session bean for the entity class.

As per the EJB 3.1 specification, business interfaces for session beans are optional. In this example, the client accessing the bean will be a local client and hence, you can use a local interface or a no-interface view to expose the bean.

Additional References:

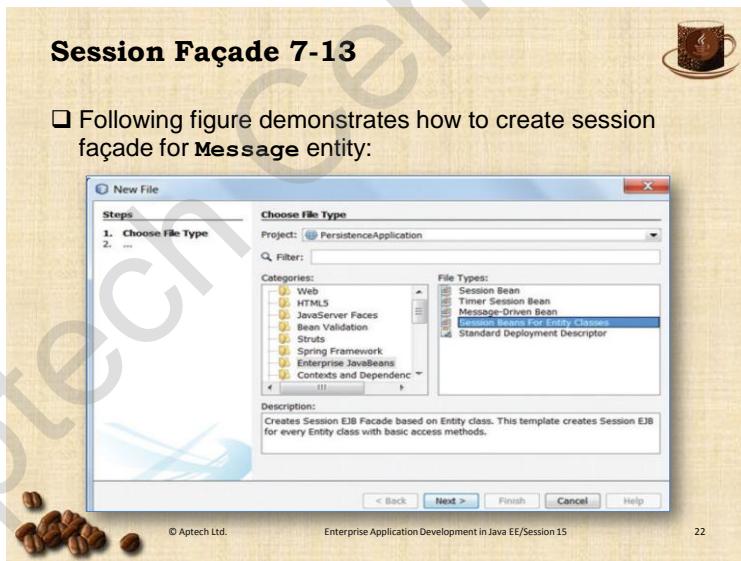
To learn more about session façade pattern, you can follow the given links:

<http://www.oracle.com/technetwork/java/sessionfacade-141285.html>

<http://www.corej2eepatterns.com/SessionFacade.htm>

Slide 22

Let us understand how to create the session façade for the Message Entity class.

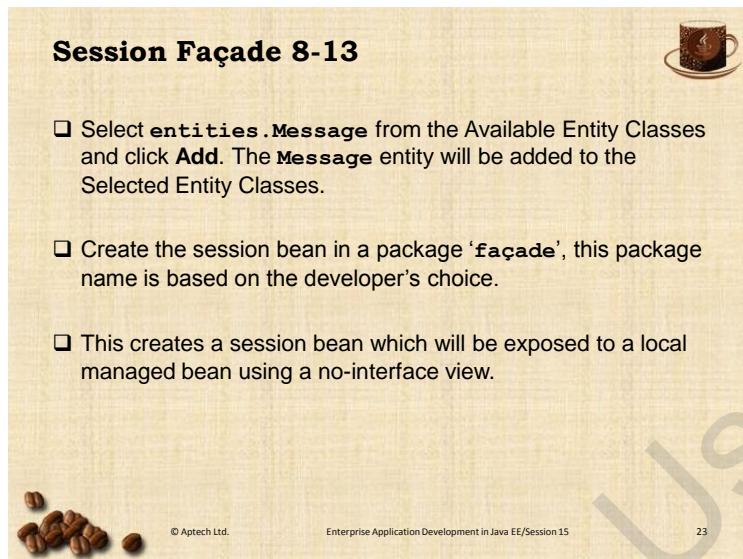


Use slide 22 to explain how developers can implement session façade for Entity classes.

According to the given wizard, developer has chosen to create a Session Beans for Entity class in PersistenceApplication. To create a session bean, right-click the project and select **New → Other → Enterprise JavaBeans → Session Beans For Entity Classes**.

Slide 23

Let us understand the steps to be followed after choosing the ‘Session Beans for Entity class’ in the wizard.



Use slide 23 to explain the steps to be followed after choosing to create ‘Session Beans for Entity classes’.

The remaining steps to create a session facade (Session bean) for the Message Entity are as follows:

1. Click **Next**. Select **entities.Message** from the **Available Entity Classes** and click **Add**. The **Message** Entity will be added to the **Selected Entity Classes**.
2. Click **Next**. Create the Session bean in a package ‘**façade**’, this package name is based on the developer’s choice.
3. Click **Finish**. This creates a Session bean which will be exposed to a local managed bean using a no-interface view.

Note, that the NetBeans IDE generated the session faced class named **MessageFacade.java** and **AbstractFacade**.

Slides 24 to 27

Let us understand the code of the abstract façade.

Session Façade 9-13



□ Following code snippet shows the code for **AbstractFacade.java**:

```
...
public abstract class AbstractFacade<T> {
    private Class<T> entityClass; @GeneratedValue(strategy = GenerationType.AUTO)
    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }
    protected abstract EntityManager getEntityManager();
    public void create(T entity) {
        getEntityManager().persist(entity);
    }
    public void edit(T entity) {
        getEntityManager().merge(entity);
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 24

Session Façade 10-13



```
public void remove(T entity) {
    getEntityManager().remove(getEntityManager().merge(entity));
}
public T find(Object id) {
    return getEntityManager().find(entityClass, id);
}
public List<T> findAll() {
    javax.persistence.criteria.CriteriaQuery cq =
        getEntityManager().getCriteriaBuilder().createQuery();
    cq.select(cq.from(entityClass));
    return getEntityManager().createQuery(cq).getResultList();
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 25

Session Façade 11-13



```
public List<T> findRange(int[] range) {
    javax.persistence.criteria.CriteriaQuery cq =
        getEntityManager().getCriteriaBuilder().createQuery();
    cq.select(cq.from(entityClass));
    javax.persistence.Query q =
        getEntityManager().createQuery(cq);
    q.setMaxResults(range[1] - range[0] + 1);
    q.setFirstResult(range[0]);
    return q.getResultList();
}

public int count() {
    javax.persistence.criteria.CriteriaQuery cq =
        getEntityManager().getCriteriaBuilder().createQuery();
    cq.select(cq.count(entityClass));
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 26

Session Façade 12-13



```
javax.persistence.criteria.Root<T> rt =  
    cq.from(entityClass);  
    cq.select(getEntityManager().getCriteriaBuilder().cou  
    nt(rt));  
    javax.persistence.Query q =  
    getEntityManager().createQuery(cq);  
    return ((Long) q.getSingleResult()).intValue();  
}  
}
```



© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 27

Use slides 24 to 27 to explain the code of `AbstractFacade` class.

When a session façade for Message Entity is created, the Netbeans IDE generates an `AbstractFacade` class which is implemented by the `MessageFacade` class.

The `AbstractFacade` class has all the methods used to manage the entities of the application. The `AbstractFacade` provides implementation of methods `edit()`, `find()`, `remove()`, `findRange()`, and `count()` which are defined to perform operations on the entities of the application.

These methods are commonly used on most of the entity classes, therefore they are already implemented by the API. The `MessageFacade` API extends this implementation and further adds functionality to it as per application requirement.

The `findAll()` method retrieves all the entities in the database and `count()` method is used to count the number of entities in the database. The `findRange()` method finds the entities within a range of values of the primary key, where the range is provided as an argument to the method.

The `MessageFacade` class extends the `AbstractFacade` class and uses `EntityManager` to manage the entities in the database.

Slide 28

Let us understand the implementation of MessageFacade.

Session Façade 13-13




Following code snippet shows the **MessageFacade** class:

```

...
@Stateless
public class MessageFacade extends AbstractFacade<Message> {
    @PersistenceContext(unitName = "PersistenceApplicationPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }
    public MessageFacade() {
        super(Message.class);
    }
}

```

The code injects the resource of the Persistence unit through an annotation and then invokes an entity manager for this context.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 28

Use slide 28 to explain the code of the **MessageFacade** class.

The **MessageFacade** here creates a new instance of **EntityManager** to manage the **Message Entity** in the application. The **Message Entity** class is defined in the persistence unit '**PersistenceApplicationPU**'. The new instance of the **EntityManager** created is associated with this persistence unit.

In-Class Question:

After you finish explaining session façade design patterns, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



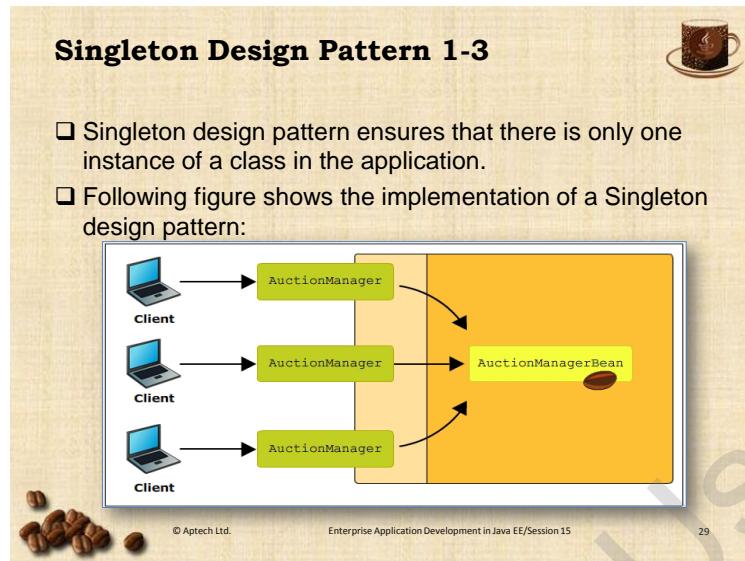
How to create Session Bean for an Entity class in NetBeans IDE?

Answer:

New → Other → Enterprise JavaBeans → Session Beans For Entity Classes.

Slide 29

Let us understand the singleton design pattern.



Use slide 29 to explain the singleton design pattern. This is a creational design pattern as it prescribes for the creation of a single instance of a Session bean in the application.

The figure as shown on slide 29 explains that the application has a single instance of `AuctionManagerBean`, which is accessed by multiple `AuctionManager` clients.

Tips:

The idea of Singleton pattern is to have only one instance of class. The usual way is just making the constructor private and either creating the object or returning the already cached instance by controlling if it has been created before.

Slide 30

Let us understand the strategies for implementing the singleton design pattern.

Singleton Design Pattern 2-3

Singleton pattern can be implemented in Java applications in one of the following ways:

- Using constructor in the Java class it can be declared **private**
- It can also be implemented as a **static** instance of a class

Singleton instance can be created according to one of the following strategies:

- Eager initialization
- Static block initialization
- Lazy initialization
- Thread safe singleton

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 30

Use slide 30 to explain different ways in which a singleton pattern can be implemented in Java applications.

A singleton pattern can be implemented in Java applications as follows:

- The constructor in the Java class can be declared **private**, this will restrict the instantiation of a class from other classes in the application.
- The singleton instance can also be implemented as a **static** instance of the class.

A singleton instance can be created through any one of the following strategies:

- **Eager initialization** – The singleton instance is created while loading the class.
- **Static block initialization** – The singleton instance is created while loading the class in a static block of code. This initialization also provides for exception handling while creating the singleton instance.
- **Lazy initialization** – When the singleton is instantiated through lazy initialization, then the instance is created in the global access method.
- **Thread safe singleton** – A thread safe singleton is created by instantiating the singleton in a synchronized block.

Additional References:

You can refer the following link to get an example on singleton pattern:

<http://viralpatel.net/blogs/java-singleton-design-pattern-tutorial-example-singleton-j2ee-design-pattern/>.

Slide 31

Let us understand the code for singleton design pattern.

Singleton Design Pattern 3-3

Following code snippet shows the instantiation of an object according to singleton pattern:

```
public class SingleInstance {
    // Declare an instance of the class
    private static SingleInstance inst = null;
    // set constructor as private
    private SingleInstance() {}
    // Define a synchronized method for creating the instance
    public static synchronized SingleInstance getInstance() {
        // check for existence of instance
        if (inst == null) {
            // create instance if it does not exist
            inst = new SingleInstance ();
        }
        return inst; // return the instance
    }
}
```

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 31

Use slide 31 to explain a sample code which implements singleton design pattern.

The code as shown on slide 31 shows a Singleton Session class. The constructor of the class is prefixed with the access modifier `private`.

The code can instantiate only one object at any instance of time. The constructor is declared as `private`, therefore external classes cannot create an instance of the class, `SingleInstance`. The `getInstance()` method creates a new instance of `SingleInstance` class only when the instance value is 'null' which implies that there is no other instance of `SingleInstance` class. It is always important to have the `getInstance()` method synchronized, so that it can remain thread safe.

However, when a singleton instance is serialized and deserialized more than once, there is a possibility of having multiple instances of the class which needs to be checked. A singleton class cannot be further inherited as its constructor is `private`.

Slide 32

Let us understand value object/transfer object pattern.

Value Object/Transfer Object Pattern

The diagram illustrates the Value Object/Transfer Object Pattern. It features a laptop icon on the left, connected by a double-headed arrow labeled "DTO" to a central blue cloud icon. The cloud is also connected by a double-headed arrow labeled "DTO" to a green rectangular box on the right. This green box contains four smaller boxes labeled "Bean". Arrows show data flow from the laptop to the cloud, from the cloud to the beans, and from the beans back to the cloud, all labeled "DTO".

The value object pattern is used while implementing communication among different entities in the application.

Data Transfer Objects are defined to minimize the number of method calls in a distributed system.

Following figure shows the data transfer object pattern:

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 32

Use slide 32 to explain value object/transfer object pattern. This is a behavioral pattern, which defines the access to various data objects in the application.

This pattern is used when the application has to minimize the number of method calls in a distributed system.

Some attributes of an entity are always accessed together in the application; such entities are combined together into an individual class. Instances of this class are termed as Data Transfer Objects (DTO). This class is a Serializable class and can be used as a return type to the business methods. Data Transfer Objects are defined to minimize the number of method calls in a distributed system. It simplifies manipulation of request and response data in the application and decouples message structures from domain layer entities.

Then, explain the figure as shown on slide 32 to explain the data transfer object pattern. Explain the figure with an example on storing Customer entity to a database. For instance, when a session bean is trying to set the values of an entity Customer, the values are obtained through the application client. In order to set the values of customer name, age, and city the application can make three method calls:

- `getName()`
- `getAge()`
- `getCity()`

Multiple method calls may lead to network latency, thus dropping the performance of the application. The Data Transfer Object pattern provides a solution by replacing these method calls by a single method call. It defines DTO in the application which replaces interfaces as the medium of communication in the application.

Tips:

The Transfer Object pattern is used when we want to pass data with multiple attributes in one pass from client to server. Transfer object is also known as Value Object. Transfer Object is a simple POJO class having getter/setter methods and is serializable so that it can be transferred over the network. The value object pattern is also called as Data Transfer Object.

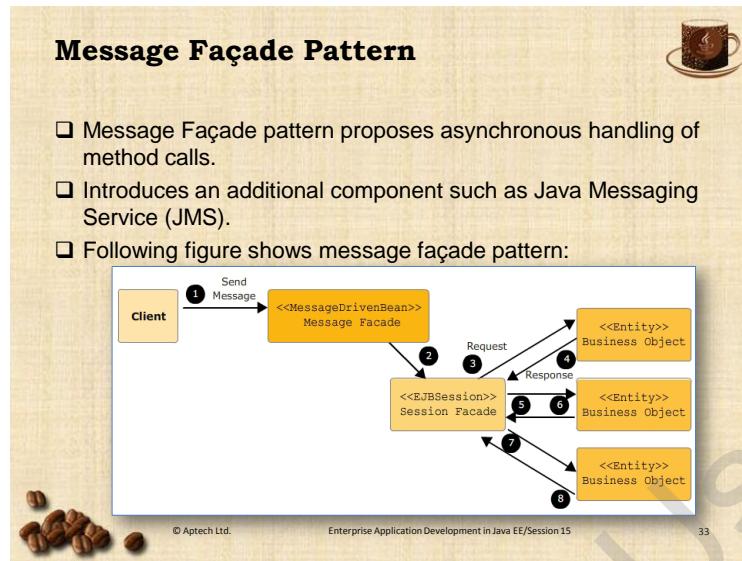
Additional References:

To prepare an example on Value Object design pattern, refer to the following link:

http://www.tutorialspoint.com/design_pattern/transfer_object_pattern.htm.

Slide 33

Let us understand message facade pattern.



Use slide 33 to explain the message façade pattern.

Message façade pattern proposes asynchronous handling of method calls. The message façade pattern introduces an additional component such as a JMS server to asynchronously handle the messages among the application components. The client sends a message to the JMS server and receives a response from the JMS server asynchronously. This mechanism avoids blocking clients as the messages are asynchronously handled by the JMS server.

Then, explain the figure as shown on slide 33 that shows the message façade pattern.

Discuss with them a scenario where the use of message façade pattern can be done while developing enterprise application.

Additional References:

To prepare yourself with an example to discuss in the class, refer to the following link:
<http://www.mcs.csueastbay.edu/~grewe/CS6320/Mat/EJB/MessageFacadePattern.htm>.

Slides 34 to 36

Let us understand the best practices to be followed while creating EJB applications.

EJB Best Practices 1-3



© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 34

- ❑ Enterprise applications developed with EJBs should have the following characteristics:
 - Remote access to the application components
 - Distributed transactions in the application domain
 - Security implementation to different components
 - Data persistence
 - Application scalability

EJB Best Practices 2-3



© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 35

- ❑ Every EJB application should have at least four files - home interface, remote interface, EJB implementation, and deployment descriptor.
- ❑ It is a good practice to write the application using JSPs and Servlets. These Web components can then be refactored and expanded to include the EJB components in the application.
- ❑ Design patterns must be used whenever possible in the applications, as they provide a reliable, tried, and tested solution for the problem.
- ❑ The complete design of the application should be available before actually implementing the application.
- ❑ Container Managed Persistence should be used whenever possible in the application as the J2EE container is capable of efficiently managing the application persistence.

EJB Best Practices 3-3

- ❑ An interface must be defined through which an EJB can be accessed. Even if the EJB is accessed locally, a local interface must be defined to access the EJB.
- ❑ Appropriate exception handling code must be written for all the exceptions that might occur in the EJB code.
- ❑ Lazy loading of database tables should be used for optimized performance of the application.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 36

Use slides 34 to 36 to explain the best practices in implementing EJB applications. Explain each practice with appropriate examples so that the students can appreciate the purpose of the stated practice.

Enterprise applications developed with EJBs should have the following characteristics:

- Remote access to the application components
- Distributed transactions in the application domain
- Security implementation to different components
- Data persistence
- Application scalability

Developers and designers should know how to use EJB in applications. Following are some of the best practices which should be incorporated while implementing EJB applications:

- Every EJB application should have at least four files home interface, remote interface, EJB implementation, and deployment descriptor. It is a good practice to write the application using JSPs and Servlets. These Web components can then be refactored and expanded to include the EJB components in the application.
- Design patterns must be used whenever possible in the applications, as they provide a reliable, tried, and tested solution for the problem.
- The complete design of the application should be available before actually implementing the application.
- Container Managed Persistence should be used whenever possible in the application as the J2EE container is capable of efficiently managing the application persistence.
- An interface must be defined through which an EJB can be accessed. Even if the EJB is accessed locally, a local interface must be defined to access the EJB.
- Appropriate exception handling code must be written for all the exceptions that might occur in the EJB code.
- Lazy loading of database tables should be used for optimized performance of the application.

Slide 37

Let us summarize the session.

Summary

- ❑ Design patterns in software are strategies which are language-independent and used for solving commonly encountered object-oriented design problems.
- ❑ The concept of design patterns was proposed by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. They are popularly known as the 'Gang-of-Four'.
- ❑ There are three primary categories of design patterns defined by GOF. They are namely, creational, structural, and behavioral patterns.
- ❑ Creational patterns enable creating objects, deciding the properties, and defining the methods of the objects based on the situation.
- ❑ Structural patterns are meant for enabling multiple classes and objects to function together. Behavioral patterns are those that determine the interaction of objects.
- ❑ There are number of patterns identified by Sun Java center for simplifying the development of enterprise applications.
- ❑ Some of the Java EE design patterns are namely, session facade, Singleton Instance, Value Object/Data Transfer Object, and Message Facade.
- ❑ The session façade manages the relationships between various business objects and provides a higher-level abstraction to the client.
- ❑ A Singleton design pattern ensures that there is only one instance of a class in the application.
- ❑ The value object pattern is used while implementing communication among different entities in the application.
- ❑ The Message Façade pattern proposes asynchronous handling of method calls.

© Aptech Ltd. Enterprise Application Development in Java EE/Session 15 37

Using slide 37, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

15.3 Post-Class Activities for Faculty

This session completes the **Enterprise Application Development in Java EE** course. Ask the students some questions related to all the topics which will help you to know the learnings taken by the students. You can solve the queries related to other sessions taught in the course.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the course. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.