UNIT 2 ENTERPRISE JAVA BEANS: ARCHITECTURE

Structure **Page Nos.**

2.0	Introduction	25
2.1	Objectives	25
2.2	Goals of Enterprise Java Beans	26
2.3	Architecture of an EJB/ Client System	26
2.4	Advantages of EJB Architecture	28
2.5	Services offered by EJB	30
2.6	Restrictions on EJB	31
2.7	Difference between Session Beans and Entity Beans	32
2.8	Choosing Entity Beans or Stateful Session Beans	33
2.9	Installing and Running Application Server for EJB	34
2.10	Summary	38
2.11	Solutions/Answers	38
2.12	Further Readings/References	42

2.0 INTRODUCTION

In the previous unit, we have learnt the basics of Enterprise java beans and different type of java beans. In this unit, we shall learn about the architecture of Enterprise java beans. “Enterprise JavaBeans” (EJBs) are distributed network aware components for developing secure, scalable, transactional and multi-user components in a J2EE environment. Enterprise JavaBeans (EJB) is suitable architecture for developing, deploying, and managing reliable enterprise applications in production environments. In this unit, we will study the benefits of using EJB architecture for enterprise applications. Enterprise application architectures have evolved through many phases. Such architectures inevitably evolve because, the underlying computer support and delivery systems have changed enormously and will continue to change in the future. With the growth of the Web and the Internet, more and more enterprise applications, including intranet and extranet applications, are now Web based. Enterprise application architectures have undergone an extensive evolution. The first generation of enterprise applications consisted of centralised mainframe applications. In the late 1980s and early 1990s, most new enterprise applications followed a two-tier, or client/server, architecture. Later, enterprise architecture evolved to a three-tier architecture and then to a Web-based architecture. The current evolutionary state is now represented by the J2EE application architecture. The J2EE architecture provides comprehensive support for two- and three-tier applications, as well as Web-based and Web services applications. Now, we will learn, about EJB architecture in detail.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- list the different goals of Enterprise java beans in the business environment;
- list the benefits offered by EJB architecture to application developers and customers;
- differentiate the services offered by EJB architecture;
- list the various kinds of restrictions for the services offered by EJB;
- differentiate the between programming style and life cycle between stateful sessions;
- make a choice between Entity beans or stateful session beans, and
- install and run application like Jboss for EJB.

2.2 GOALS OF JAVA ENTERPRISE BEANS

The EJB Specifications try to meet several goals which are described as following:

- EJB is designed to make it easy for developers to create applications, freeing them from low-level system details of managing transactions, threads, load balancing, and so on. Application developers can concentrate on business logic and leave the details of managing the data processing to the framework. For specialised applications, though, it's to customise these lower-level services.

- The EJB Specifications define the major structures of the EJB framework, and then specifically define the contracts between them. The responsibilities of the client, the server, and the individual components are all clearly spelled out. A developer creating an Enterprise JavaBean component has a very different role from someone creating an EJB-compliant server, and the specification describes the responsibilities of each.

- EJB aims to be the standard way for client/server applications to be built in the Java language. Just as the original JavaBeans (or Delphi component etc.) from different vendors can be combined to produce a custom client, EJB server components from different vendors can be combined to produce a custom server. EJB components, being Java classes, will of course run in any EJB-compliant server without recompilation. This is a benefit that platform-specific solutions cannot offer.

- Finally, the EJB is compatible with and uses other Java APIs, can interoperate with non-Java applications, and is compatible with CORBA.

2.3 ARCHITECTURE OF AN EJB/ CLIENT SYSTEM

There are various architectures of EJB (J2EE) available which are used for various purposes. J2EE is a standard architecture specifically oriented to the development and deployment of enterprise Web-oriented applications that use Java programming language. ISVs and enterprises can use the J2EE architecture for only not the development and deployment of intranet applications, thus effectively replacing the two-tier and three-tier models, but also for the development of Internet applications, effectively replacing the cgi-bin-based approach. The J2EE architecture provides a flexible distribution and tiering model that allows enterprises to construct their applications in the most suitable manner.

Now, let us understand how an EJB client/server system operates and the underlying architecture of EJB, we need to understand the basic parts of an EJB system: *the EJB component, the EJB container, and the EJB object.*

The Enterprise JavaBeans Component

An Enterprise JavaBean is a component, just like a traditional JavaBean. Enterprise JavaBeans execute within an EJB container, which in turn executes within an EJB server. Any server that can host an EJB container and provide it with the necessary services can be an EJB server. (Hence, many existing servers are being extended to be EJB servers.) An EJB component is the type of EJB class most likely to be considered an “Enterprise JavaBean”. It’s a Java class, written by an EJB developer, that implements business logic. All the other classes in the EJB system either support client access to or provide services (like persistence, and so on) to EJB component classes.

The Enterprise JavaBeans Container

The EJB container is where the EJB component “lives”. The EJB container provides services such as transaction and resource management, versioning, scalability, mobility,

persistence, and security to the EJB components it contains. Since the EJB container handles all these functions, the EJB component developer can concentrate on business rules, and leave database manipulation and other such fine details to the container. For example, if a single EJB component decides that the current transaction should be aborted, it simply tells its container (container is responsible for performing all rollbacks, or doing whatever is necessary to cancel a transaction in progress). Multiple EJB component instances typically exist inside a single EJB container.

The EJB Object and the Remote Interface

Client programs execute methods on remote EJBs by way of an EJB object. The EJB object implements the "remote interface" of the EJB component on the server. The remote interface represents the "business" methods of the EJB component. The remote interface does the actual, useful work of an EJB object, such as creating an order form or deferring a patient to a specialist. EJB objects and EJB components are separate classes, though from the outside (i.e., by looking at their interfaces), they look identical. This is because, they both implement the same interface (the EJB component's remote interface), but they do very different things. An EJB component runs on the server in an EJB container and implements the business logic. *The EJB object runs on the client and remotely executes the EJB component's methods.*

Let us consider an analogy to understand the concept of EJB. Assume your VCD player is an EJB component. The EJB object is then analogous to your remote control: both the VCD and the remote control have the same buttons on the front, but they perform different functions. By pushing the Rewind button on your VCD player's remote control is equivalent to pushing the Rewind button on the VCD player itself, even though it's the VCD player -- and not the remote -- that actually rewinds a tape.

Working of EJB

The actual implementation of an EJB object is created by a code generation tool that comes with the EJB container. The EJB object's interface is the EJB component's remote interface. The EJB object (created by the container and tools associated with the container) and the EJB component (created by the EJB developer) implement the same remote interface. To the client, an EJB object looks just like an object from the application domain -- an order form, for example. But the EJB object is just a stand-in for the actual EJB, running on the server inside an EJB container. When the client calls a method on an EJB object, the EJB object method communicates with the remote EJB container, requesting that the same method be called, on the appropriate (remote) EJB, with the same arguments, on the client's behalf. This is explained with the help of the *Figure1*. This is the core concept behind how an EJB client/server system works.

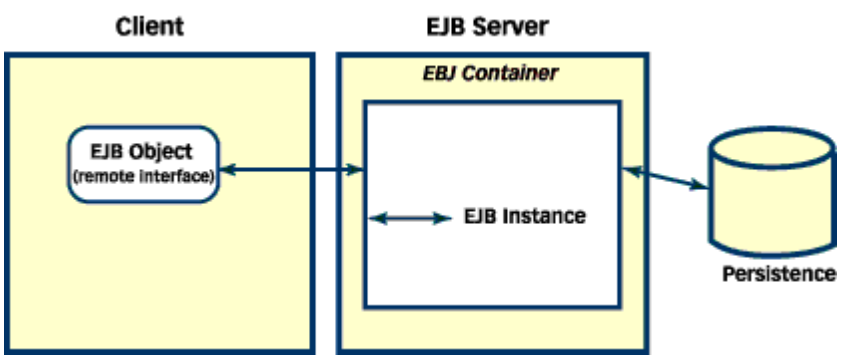

Figure 1: Enterprise JavaBeans in the Client and the Server

2.4 ADVANTAGES OF EJB ARCHITECTURE

Benefits to Application Developers:

The EJB component architecture is the backbone of the J2EE platform. The EJB architecture provides the following benefits to the application developer:

- **Simplicity:** It is easier to develop an enterprise application with the EJB architecture than without it. Since, EJB architecture helps the application developer access and use enterprise services with minimal effort and time, writing an enterprise bean is almost as simple as writing a Java class. The application developer does not have to be concerned with system-level issues, such as security, transactions, multithreading, security protocols, distributed programming, connection resource pooling, and so forth. As a result, the application developer can concentrate on the business logic for domain-specific applications.

- **Application Portability:** An EJB application can be deployed on any J2EE-compliant server. This means that the application developer can sell the application to any customers who use a J2EE-compliant server. This also means that enterprises are not locked into a particular application server vendor. Instead, they can choose the “best-of-breed” application server that meets their requirements.

- **Component Reusability:** An EJB application consists of enterprise bean components. Each enterprise bean is a reusable building block. There are two essential ways to reuse an enterprise bean:

  1) An enterprise bean not yet deployed can be reused at application development time by being included in several applications. The bean can be customised for each application without requiring changes, or even access, to its source code.

  2) Other applications can reuse an enterprise bean that is already deployed in a customer's operational environment, by making calls to its client-view interfaces. Multiple applications can make calls to the deployed bean.

  In addition, the business logic of enterprise beans can be reused through Java subclassing of the enterprise bean class.

- **Ability to Build Complex Applications:** The EJB architecture simplifies building complex enterprise applications. These EJB applications are built by a team of developers and evolve over time. The component-based EJB architecture is well suited to the development and maintenance of complex enterprise applications. With its clear definition of roles and well-defined interfaces, the EJB architecture promotes and supports team-based development and lessens the demands on individual developers.

- **Separation of Business Logic from Presentation Logic:** An enterprise bean typically encapsulates a business process or a business entity (an object representing enterprise business data), making it independent of the presentation logic. The business programmer need not worry about formatting the output; the Web page designer developing the Web page need be concerned only with the output data that will be passed to the Web page. In addition, this separation makes it possible to develop multiple presentation logic for the same business process or to change the presentation logic of a business process without needing to modify the code that implements the business process.

- **Easy Development of Web Services:** The Web services features of the EJB architecture provide an easy way for Java developers to develop and access Web services. Java developers do not need to bother about the complex details of Web services description formats and XML-based wire protocols but instead can program at the familiar level of enterprise bean components, Java interfaces and data types. The tools provided by the container manage the mapping to the Web services standards.

- **Deployment in many Operating Environments**— The goal of any software development company is to sell an application to many customers. Since, each customer has a unique operational environment, the application typically needs to be customised at deployment time to each operational environment, including different database schemas.

    1) The EJB architecture allows the bean developer to separate the common application business logic from the customisation logic performed at deployment.

    2) The EJB architecture allows an entity bean to be bound to different database schemas. This persistence binding is done at deployment time. The application developer can write a code that is not limited to a single type of database management system (DBMS) or database schema.

    3) The EJB architecture facilitates the deployment of an application by establishing deployment standards, such as those for data source lookup, other application dependencies, security configuration, and so forth. The standards enable the use of deployment tools. The standards and tools remove much of the possibility of miscommunication between the developer and the deployer.

- **Distributed Deployment:** The EJB architecture makes it possible for applications to be deployed in a distributed manner across multiple servers on a network. The bean developer does not have to be aware of the deployment topology when developing enterprise beans but rather writes the same code whether the client of an enterprise bean is on the same machine or a different one.

- **Application Interoperability:** The EJB architecture makes it easier to integrate applications from different vendors. The enterprise bean's client-view interface serves as a well-defined integration point between applications.

- **Integration with non-Java Systems:** The related J2EE APIs, such as the J2EE Connector specification and the Java Message Service (JMS) specification, and J2EE Web services technologies, such as the Java API for XML-based RPC (JAX-RPC), make it possible to integrate enterprise bean applications with various non-Java applications, such as ERP systems or mainframe applications, in a standard way.

- **Educational Resources and Development Tools:** Since, the EJB architecture is an industry wide standard, the EJB application developer benefits from a growing body of educational resources on how to build EJB applications. More importantly, powerful application development tools available from leading tool vendors simplify the development and maintenance of EJB applications.

Benefits to Customers

A customer’s perspective on EJB architecture is different from that of the application developer. The EJB architecture provides the following benefits to the customer:

- **Choice of Server:** Because the EJB architecture is an industry wide standard and is part of the J2EE platform, customer organisations have a wide choice of J2EE-compliant servers. Customers can select a product that meets their needs in terms of scalability, integration capabilities with other systems, security protocols, price, and

so forth. Customers are not locked in to a specific vendor's product. Should their needs change, customers can easily redeploy an EJB application in a server from a different vendor.

- **Facilitation of Application Management:** Because the EJB architecture provides a standardised environment, server vendors have the required motivation to develop application management tools to enhance their products. As a result, sophisticated application management tools provided with the EJB container allow the customer's IT department to perform such functions as starting and stopping the application, allocating system resources to the application, and monitoring security violations, among others.

- **Integration with a Customer's Existing Applications and Data:** The EJB architecture and the other related J2EE APIs simplify and standardise the integration of EJB applications with any non-Java applications and systems at the customer operational environment. For example, a customer does not have to change an existing database schema to fit an application. Instead, an EJB application can be made to fit the existing database schema when it is deployed.

- **Integration with Enterprise Applications of Customers, Partners, and Suppliers:** The interoperability and Web services features of the EJB architecture allows EJB-based applications to be exposed as Web services to other enterprises. This means that enterprises have a single, consistent technology for developing intranet, Internet, and e-business applications.

- **Application Security:** The EJB architecture shifts most of the responsibility for an application's security from the application developer to the server vendor, system administrator, and the deployer. The people performing these roles are more qualified than the application developer to secure the application. This leads to better security of operational applications.

2.5 SERVICES OFFERED BY EJB

As we have learnt earlier, in J2EE all the components run inside their own containers. JSP, Servlets and JavaBeans and EJBs have their own web container. The container of EJB provides certain built-in services to EJBs, which is used by EJBs to perform different functions. The services that EJB container provides are:

- Component Pooling
- Resource Management
- Transaction Management
- Security
- Persistence
- Handling of multiple clients

i) Component Pooling

The EJB container handles the pooling of EJB components. If, there are no requests for a particular EJB then the container will probably contain zero or one instance of that component in the memory. If, the need arises then it will increase component instances to satisfy all incoming requests. Then again, if, the number of requests decrease, the container will decrease the component instances in the pool. The most important thing is that the client is absolutely unaware of this component pooling which the container handles.

ii) Resource Management

The container is also responsible for maintaining database connection pools. It provides us a standard way of obtaining and returning database connections. The container also manages EJB environment references and references to other EJBs. The container manages the following types of resources and makes them available to EJBs:

- JDBC 2.0 Data Sources
- JavaMail Sessions
- JMS Queues and Topics
- URL Resources
- Legacy Enterprise Systems via J2EE Connector Architecture

iii) Transaction Management

This is the most important functionality served by the container. A transaction is a single unit of work, composed of one or more steps. If, all the steps are successfully executed, then, the transaction is committed otherwise it is rolled back. There are different types of transactions and it is absolutely unimaginable not to use transactions in today's business environments.

iv) Security

The EJB container provides it's own authentication and authorisation control, allowing only specific clients to interact with the business process. Programmers are not required to create a security architecture of their own, they are provided with a built-in system, all they have to do is to use it.

v) Persistence

Persistence is defined as saving the data or state of the EJB on non J-C volatile media. The container, if, desired can also maintain persistent data of the application. The container is then responsible for retrieving and saving the data for programmers, while taking care of concurrent access from multiple clients and not corrupting the data.

vi) Handling of Multiple Clients

The EJB container handles multiple clients of different types. A JSP based thin client can interact with EJBs with same ease as that of GUI based thick client. The container is smart enough to allow even non-Java clients like COM based applications to interact with the EJB system.

2.6 RESTRICTIONS ON EJB

Enterprise Java Beans have several restrictions, which they must adhere to:

1. They cannot create or manage threads.
2. They cannot access threads using the java.io package.
3. They cannot operate directly with sockets.
4. They cannot load native libraries.
5. They cannot use the AWT to interact with the user.
6. They can only pass objects and values which are compatible with RMI/IIOP.
7. They must supply a public no argument constructor.
8. Methods cannot be static or final.
9. There are more minor restrictions.

But following can be done with EJB:

1. Subclass another class.
2. Interfaces can extend other interfaces, which are descendants of EJBObject or EJBHome.
3. Helper methods can pass any kind of parameters or return types within the EJB.
4. Helper methods can be any kind of visibility.

2.7 DIFFERENCE BETWEEN SESSION BEANS AND ENTITY BEANS

To understand whether a business process or business entity should be implemented as a stateful session bean or an entity bean, it is important to understand the life-cycle and programming differences between them. These differences pertain principally to object sharing, object state, transactions, and container failure and object recovery.
Object sharing pertains to entity objects. Only a single client can use a stateful session bean, but multiple clients can share an entity object among themselves.

The container typically maintains a stateful session bean's object state in the main memory, even across transactions, although the container may swap that state to secondary storage when deactivating the session bean. The object state of an entity bean is typically maintained in a database, although the container may cache the state in memory during a transaction or even across transactions. Other, possibly non-EJB-based, programs can access the state of an entity object that is externalised in the database. For example, a program can run an SQL query directly against the database storing the state of entity objects. In contrast, the state of a stateful session object is accessible only to the session object itself and the container.

The state of an entity object typically changes from within a transaction. Sinces, its state changes transactionally, the container can recover the state of an entity bean should the transaction fail. The container does not maintain the state of a session object transactionally. However, the bean developer may instruct the container to notify the stateful session objects of the transaction boundaries and transaction outcome. These notifications allow the session bean developer to synchronise manually the session object's state with the transactions. For example, the stateful session bean object that caches changed data in its instance variables may use the notification to write the cached data to a database before the transaction manager commits the transaction.

Session objects are not recoverable; that is, they are not guaranteed to survive a container failure and restart. If, a client has held a reference to a session object, that reference becomes invalid after a container failure. (Some containers implement session beans as recoverable objects, but this is not an EJB specification requirement.) An entity object, on the other hand, survives a failure and restart of its container. If, a client holds a reference to the entity object prior to the container failure, the client can continue to use this reference after the container restarts.

The *Table 1* depicts the significant differences in the life cycles of a stateful session bean and an entity bean.

Table 1: Entity Beans and Stateful Session Beans: Life-Cycle Differences

Functional Area	Stateful Session Bean	Entity Bean
Object state	Maintained by the container in the main memory across transactions. Swapped to secondary storage when deactivated.	Maintained in the database or other resource manager. Typically cached in the memory in a transaction.
Object sharing	A session object can be used by only one client.	An entity object can be shared by multiple clients. A client may pass an object reference to another client.
State externalisation	The container internally maintains the session object's state. The state is inaccessible to other programs.	The entity object's state is typically stored in a database. Other programs, such as an SQL query, can access the state in the database.
Transactions	The state of a session object can be synchronised with a transaction but is not recoverable.	The state of an entity object is typically changed transactionally and is recoverable.
Failure recovery	A session object is not guaranteed to survive failure and restart of its container. The references to session objects held by a client becomes invalid after the failure.	An entity object survives the failure and the restart of its container. A client can continue using the references to the entity objects after the container restarts.

2.8 CHOOSING ENTITY BEANS OR STATEFUL SESSION BEANS

Now, we will learn when and where we should use entity beans or stateful session beans. The architect of the application chooses how to map the business entities and processes to enterprise beans. No prescriptive rules dictate whether a stateful session bean or an entity bean should be used for a component: Different designers may map business entities and processes to enterprise beans differently.

We can also combine the use of session beans and entity beans to accomplish a business task. For example, we may have a session bean represent an ATM withdrawal that invokes an entity bean to represent the account.

The following guidelines outline the recommended mapping of business entities and processes to entity and session beans. The guidelines reflect the life-cycle differences between the session and entity objects.

- A bean developer typically implements a business entity as an entity bean.
- A bean developer typically implements a conversational business process as a stateful session bean. For example, developers implement the logic of most Web application sessions as session beans.
- A bean developer typically implements, as an entity bean a collaborative business process: a business process with multiple actors. The entity object's state represents the intermediate steps of a business process that consists of multiple steps. For example, an entity object's state may record the changing information—state—on a loan application as it moves through the steps of the loan-approval process. The object's state may record that the account representative entered the information on the loan application, the loan officer reviewed the application, and application approval is still waiting on a credit report.
- If, it is necessary for any reason to save the intermediate state of a business process in a database, a bean developer implements the business process as an entity bean. Often, the saved state itself can be considered a business entity. For example, many e-commerce Web applications use the concept of a shopping cart, which stores the

items that the customer has selected but not yet checked out. The state of the shopping cart can be considered to be the state of the customer shopping business process. If, it is desirable that the shopping process span extended time periods and multiple Web sessions, the bean developer should implement the shopping cart as an entity bean. In contrast, if the shopping process is limited to a single Web session, the bean developer can implement the shopping cart as a stateful session bean.

2.9 INSTALLING AND RUNNING APPLICATION SERVER FOR EJB

Depending on the requirements of the organisation or company, one may select an appropriate Application server. If organisation's business processes span hundreds or thousands of different computers and servers then, one can select good application servers like BEA Web Logic, IBM Web Sphere and Oracle 9i Application Server etc. which are licensed servers. But, on the other hand, if, the organisation is really small but still wants to use EJBs due to future scalability requirements then, there are quite a few EJB application servers from free and open source to the ones, which are fast, but with reasonably low license fees. Examples of application servers, which are popular in small organisations, include **JBoss** and **Orion** Application Server.

An application server is a conglomeration of software services that provide a runtime environment for any number of containers, as shown in the *Figure 2*. A typical J2EE application server, such as WebLogic, WebSphere, JBoss, and Sun's J2EE Reference Server, houses a multitude of containers. WebLogic, for example, supports an EJB container and a servlet container.

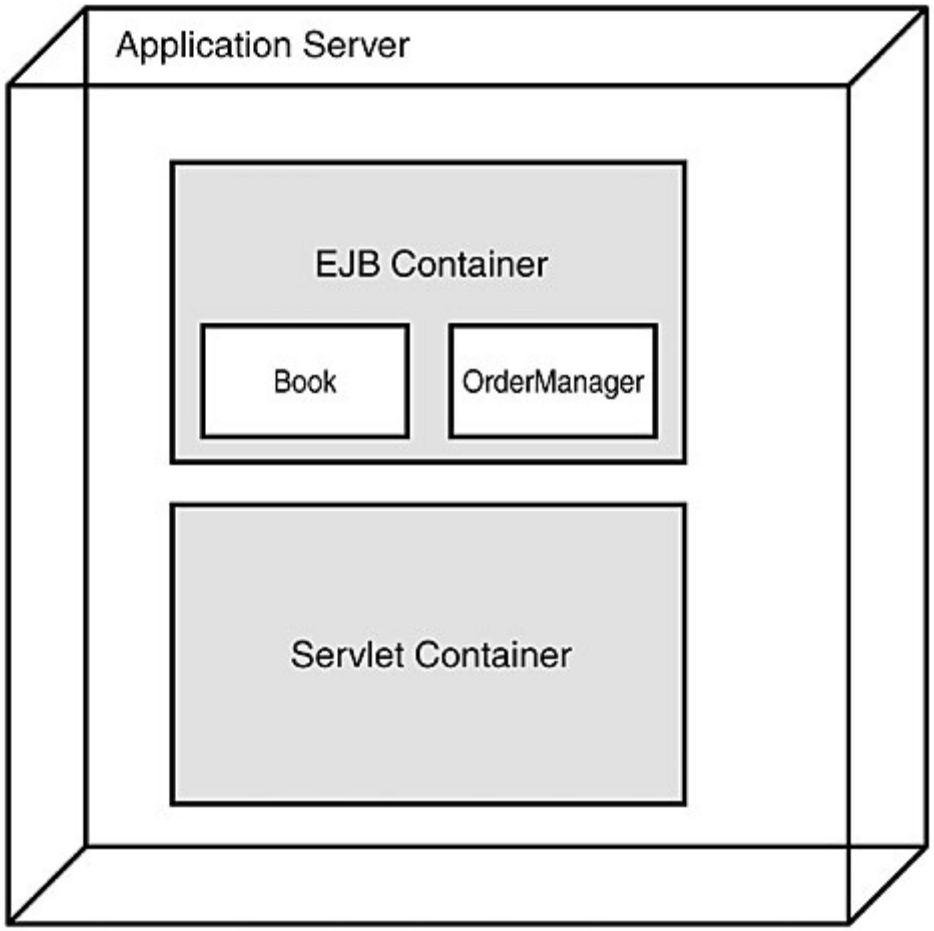

Figure 2: Architecture of EJB container

The EJB container provides basic services, including transactions, life-cycle management, and security, to the EJBs deployed into it. By shouldering much of this burdensome lower-level functionality, the EJB container significantly reduces the responsibilities of the EJBs deployed into it. Because EJBs no longer contain the code to provide these fundamental behaviours, EJB developers are free to concentrate on writing code that solves business problems instead of computer science problems.

Every application server vendor has its own way of deploying EJBs. They all share some common traits, however, that are illustrated in *Figure 3* and described here:

- An EJB's class (or sometimes its source code) files and its deployment descriptor are placed into an archive, which is typically a JAR file. Deployment descriptors are described in more depth in Part II, "Reference".

- A deployment tool of some sort creates a deployable archive file (typically, but not always, a JAR) from the contents of the archive created in Step 1.

- The deployable archive file is deployed into the EJB container by editing the container's configuration file or by running an administrative interface program.

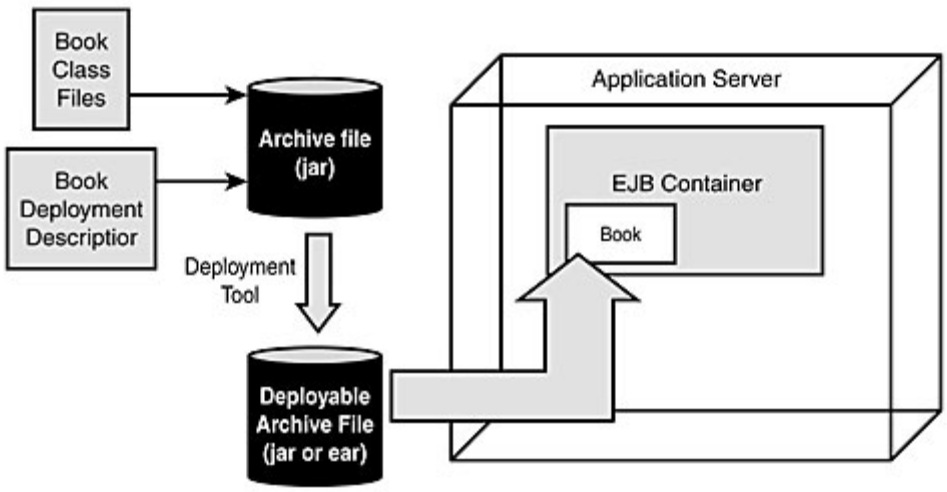

Figure 3: EJBs are typically bundled into archive files, processed by a deployment tool, and then deployed into the EJB container

There are many free application servers like Sun's J2EE Reference Application Server, which is available free at http://www.javasoft.com. Or Jboss, which may be downloaded JBoss from JBoss web site: www.jboss.org. Current stable version is 2.4.3. Download it from their web site. Once you have downloaded it, unzip the JBoss zip file into some directory e.g. C:\JBoss. The directory structure should be something like the following:

```
C:\JBoss
 admin
 bin
 client
 conf
  db
```

```
deploy
lib
log
tmp
```

Now, to start JBoss with default configuration go to JBoss/bin directory and run the following command at the DOS prompt :

```
C:\JBoss\bin>run
```

run.bat is a batch file which starts the JBoss Server. Once JBoss Server starts, you should see huge lines of text appearing on your command prompt screen. These lines show that JBoss Server is starting. Once JBoss startup is complete you should see a message like following one on your screen :

```
[Default] JBoss 2.4.3 Started in 0m:11s
```

Now, we have successfully installed and run JBoss on your system. To stop JBoss, simply press Ctrl + C on the command prompt and JBoss will stop, after displaying huge lines of text.

The client for our EJB will be a JSP page / Java Servelet running in a separate Tomcat Server. We have already learnt in an earlier Block 1, how to create and install Tomcat server for running JSP page or Servlet

Configuring and Running Tomcat :

Create a new folder under the main C:\ drive and name it “Projects”. Now, create a new sub-folder in the C:\Projects folder and name it "TomcatJBoss". The directory structure should look like the following:

```
C:\Projects
    TomcatJBoss
```

Now, open conf/Server.xml file from within the Tomcat directory where you have installed it. By default this location will be :

```
C:\Program Files\Apache Tomcat 4.0\conf\server.xml
```

Somewhere in the middle where you can see multiple <Context> tags, add following lines between other <Context> tags :

```
<!-- Tomcat JBoss Context -->
<Context path=”/jboss” docBase=“C:\Projects\TomcatJBoss\” debug=“0” reloadable= “true” />
```

Now, save Server.xml file. Go to Start -> Programs -> Apache Tomcat 4.0 -> Start Tomcat, to start Tomcat Server. If everything has been setup correctly, you should see the following message on your command prompt:

Starting service Tomcat-Standalone

Apache Tomcat/4.0.1

Note: Creating C:\Projects\TomcatJBoss folder and setting up new /jboss context in Tomcat is NOT required as far as accessing EJBs is concerned. We are doing it only to put our JSP client in a separate folder so as not to intermingle it with your other projects.

Check Your Progress 1

1) State True or False

a) EJB can manage threads. T ☐ F ☐

F

b) In EJB Methods can be static or final. T ☐ F ☐

c) A transaction is a single work, composed of one or more steps. T ☐ F ☐

d) The EJB container handles multiple clients of different types. T ☐ F ☐

2) Describe the layered architecture of EJB and explain all its components briefly.

……..…

…..……..…

……

3) Explain the different benefits of EJB architecture to Application Developers and Customers.

……..…

…..……..…

……

4) How does the EJB container assist in transaction management and persistence?

……..…

…..……..…

……

5) Explain the different types of restrictions on EJB.

……..…

…..……..…

……

6) How does the Session Bean differ from the Entity Bean in terms of object sharing, object state and failure recovery?

……..…

…..……..…

……

7) What criteria should a developer keep in mind while choosing between a session bean and an entity bean?

……..…

…..……..…

……

2.10 SUMMARY

In this unit, we have learnt basic architecture of EJB and the various kind of benefits offered by it to application developers and customers. We have also highlighted services offered by EJB, restriction emposed on EJB. The difference between session beans and entitiy beans and choasing between entity beans and session beans. Finally we have discussed how to install and run application server for EJB.

2.11 SOLUTIONS/ ANSWERS

Check Your Progress 1

1) True/ False

a) False

b) False

c) True

d) True

Explanatory Answers

2) EJB is a layered architecture consisting of:

- Enterprise bean component which contains methods that implements the business logic
- EJB container
- EJB server, which contains the EJB container
- EJB object.

The Enterprise Java Beans Component

An Enterprise JavaBean is a component, just like a traditional JavaBean. Enterprise JavaBeans execute within an EJB container, which in turn executes within an EJB server. Any server that can host an EJB container and provide it with the necessary services can be an EJB server. EJB is a Java class, written by an EJB developer, that implements business logic. All the other classes in the EJB system either support client access to or provide services (like persistence, and so on) to EJB component classes.

The Enterprise Java Beans Container

The EJB container acts as an interface between an Enterprise bean and clients. The client communicates with the Enterprise bean through the remote and home interfaces provided by the customer. The EJB container is where the EJB component "lives." Since the EJB container handles all these functions, the EJB component developer can concentrate on business rules, and leave database manipulation and other such fine details to the container. The EJB container provides services such as security, transaction and resource management, versioning, scalability, mobility, persistence, and security to the EJB components it contains.

EJB Server

The EJB server provides some low level services like network connectivity to the container. In addition to this, it also provides the following services:

- Instance Passivation
- Instance pooling

- Database Connection Pooling
- Preached Instances

The EJB Object and the Remote Interface

Client programs execute methods on remote EJBs by way of an EJB object. The EJB object implements the "remote interface" of the EJB component on the server. The remote interface represents the "business" methods of the EJB component. The remote interface does the actual, useful work of an EJB object, such as creating an order form or referring a patient to a specialist.

EJB objects and EJB components are separate classes, though from the outside (i.e., by looking at their interfaces), they look identical. This is because, they both implement the same interface (the EJB component's remote interface), but they do very different things. An EJB component runs on the server in an EJB container and implements the business logic. *The EJB object runs on the client and remotely executes the EJB component's methods.*

3) The EJB component architecture is the backbone of the J2EE platform. The EJB architecture provides the following benefits to the application developer:

   - **Simplicity:** It is easier to develop an enterprise application with EJB architecture than without it. Since, EJB architecture helps the application developer access and use enterprise services with minimal effort and time, writing an enterprise bean is almost as simple as writing a Java class.

   - **Application Portability:** An EJB application can be deployed on any J2EE-compliant server. Application developer can choose the "best-of-breed" application server that meets their requirements.

   - **Component Reusability:** An EJB application consists of enterprise bean components and each of the enterprise Bean is a reusable building block. Business logic of enterprise beans can be reused through Java sub classing of the enterprise bean class.

   - **Ability to Build Complex Applications:** EJB architecture simplifies building complex enterprise applications. The component-based EJB architecture is well suited to the development and maintenance of complex enterprise applications.

   - **Separation of Business Logic from Presentation Logic:** An enterprise bean typically encapsulates a business process or a business entity (an object representing enterprise business data), making it independent of the presentation logic.

   - **Easy development of Web Services:** The Web services features of the EJB architecture provide an easy way for Java developers to develop and access Web services. The tools provided by the container manage the mapping to the Web services standards.

   - **Deployment in many Operating Environments:** The EJB architecture allows the bean developer to separate common application business logic from the customisation logic performed at deployment to me. The EJB architecture allows an entity bean to be bound to different database schemas. The EJB architecture facilitates the deployment of an application by establishing deployment standards, such as those for data source lookup, other application dependencies, security configuration, and so forth.

- **Distributed Deployment:** The EJB architecture makes it possible for applications to be deployed in a distributed manner across multiple servers on a network.
- **Application Interoperability:** The EJB architecture makes it easier to integrate applications from different vendors. The enterprise bean's client-view interface serves as a well-defined integration point between applications.
- **Integration with non-Java systems:** The related J2EE APIs, such as the J2EE Connector specification and the Java Message Service (JMS) specification, and J2EE Web services technologies, such as the Java API for XML-based RPC (JAX-RPC), make it possible to integrate enterprise bean applications with various non-Java applications, such as ERP systems or mainframe applications, in a standard way.
- **Educational Resources and Development Tools:** Because EJB architecture is an industry wide standard, the EJB application developer benefits from a growing body of educational resources on how to build EJB applications.

Benefits to Customers

A customer's perspective on EJB architecture is different from that of the application developer. The EJB architecture provides the following benefits to the customer:

- **Choice of Server:** Because the EJB architecture is an industry wide standard and is part of the J2EE platform, customer organisations have a wide choice of J2EE-compliant servers. Customers can select a product that meets their needs in terms of scalability, integration capabilities with other systems, security protocols, price, and so forth.
- **Facilitation of Application Management:** Sophisticated application management tools provided with the EJB container allow the customer's IT department to perform such functions as starting and stopping the application, allocating system resources to the application, and monitoring security violations, among others.
- **Integration with a Customer's Existing Applications and Data:** EJB architecture and the other related J2EE APIs simplify and standardize the integration of EJB applications with any non-Java applications and systems at the customer operational environment.
- **Integration with Enterprise Applications of Customers, Partners, and Suppliers:** The interoperability and Web services features of the EJB architecture allow EJB-based applications to be exposed as Web services to other enterprises.
- **Application Security:** EJB architecture shifts most of the responsibility for an application's security from the application developer to the server vendor, system administrator, and the deployer.

4) EJB container offers various kinds of services. Transaction Management is the most important functionality served by the container. The container provides the support according to the transaction specified and we can specify following types of transaction support for the bean:

- The bean manages its own transaction
- The bean doesn't support any transaction.
- When the client invokes the bean's method and if, the client has a transaction in progress, then the bean runs within the client's transaction. Otherwise, the container starts a new transaction for the bean.

- The bean must always start a new transaction, even if a transaction is open.
- The bean requires the client to have a transaction open before the client invokes the bean's method.

Persistence is another important function, which is supported by the EJB container. Persistence is the permanent storage of state of an object in a non-volatile media. This allows an object to be accessed at any time, without having to recreate the object every time it is required. The container if desired can also maintain persistent data of the application.

5) Enterprise Java Beans have several restrictions, which they must adhere to:

1. They cannot create or manage threads.
2. They cannot access threads using the java.io package.
3. They cannot operate directly with sockets.
4. They cannot load native libraries.
5. They cannot use the AWT to interact with the user.
6. They can only pass objects and values, which are compatible with RMI/IIOP.
7. They must supply a public no argument constructor.
8. Methods cannot be static or final.
9. There are more minor restrictions.

6) Session Beans and Entity beans differ in respect to object sharing, object state, transactions and container failure and object recovery.

Object Sharing: A session object can be used by only one client whereas, an entity object can be shared by multiple clients. A client may pass an object reference to another client.

Object State: Object state of stateful session bean is maintained by the container in the main memory across transactions and swapped to secondary storage when deactivated whereas, the object state of an entity bean is maintained in the database or other resource manager and typically cached in the memory in a transaction.

Failure Recovery: A session object is not guaranteed to survive failure and restart of its container. The references to session objects held by a client become invalid after the failure whereas, an entity object survives the failure and the restart of its container. A client can continue using the references to the entity objects after the container restarts.

7) There could be various guidelines, which can assist application developer choose between a session bean and entity bean, which are described below:

- A bean developer may typically implement a business entity as an entity bean.
- A bean developer may typically implement a conversational business process as a stateful session bean. For example, developers implement the logic of most Web application sessions as session beans.
- A bean developer typically implements as an entity bean a collaborative business process: a business process with multiple actors.
- If, it is necessary for any reason to save the intermediate state of a business process in a database, a bean developer implements the business process as an entity bean. Often, the saved state itself can be considered a business entity.

2.12 FURTHER READINGS/REFERENCES

- Paco Gomez and Peter Zadronzy, *Professional Java 2 Enterprise Edition with BEA Weblogic Server,* WROX Press Ltd
- Budi Kurniawan, *Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions,* New Riders Publishing
- Richard Monson-Haefel, *Enterprise JavaBeans (3rd Edition),* O'Reilly
- Vlada Matena, Sanjeev Krishnan, Linda DeMichiel, Beth Stearns, *Applying Enterprise JavaBeans™:Component-Based Development for the J2EE™ Platform,* Second Edition, Pearson Education
- Robert Englander, *Developing Java Beans,* O'Reilly Media

Reference websites:

- www.j2eeolympus.com
- www.phptr.com
- www.samspublishing.com
- www.oreilly.com
- www.roseindia.net
- www.caucho.com
- www.tutorialized.com
- www.stardeveloper.com