Containers .............................................................................................................................................. 2
1. WHAT IS A J2EE CONTAINER? ........................................................................................................... 2
2. WHAT IS ENTERPRISE J2EE EJB SERVER OR CONTAINER? ......................................................... 2
3. WHAT IS ENTERPRISE JAVA BEANS (EJB) IN J2EE? WHAT IS ADVANTAGE OF USING EJB? .............. 3
4. CAN WE RUN EJB IN WEB SERVER LIKE TOMCAT? ........................................................................... 3
5. WHAT WERE SHORTCOMINGS OF EJB 2.0? .................................................................................... 3
6. WHAT IS ADVANTAGE OF PUTTING BUSINESS LOGIC IN EJB OVER STORED PROCEDURE? ......... 3
7. WHAT ARE THE SERVICES PROVIDED BY EJB CONTAINER? ......................................................... 3
8. WHAT IS TWO PHASE COMMIT? HOW DO YOU MANAGE DISTRIBUTED TRANSACTION IN EJB? ......... 3
9. WHAT IS MAIN DIFFERENCE BETWEEN EJB 2.0 AND EJB 3.0? ................................................. 4
10. CAN YOU NAME FEW ANNOTATION USED IN EJB 3.0 AND WHAT DOES HE DO? .................... 4
11. HOW TO CALL AN EJB 3.0 BEAN FROM SERVLET OR JSP? ...................................................... 4
What is a J2EE Container?

- A Web application runs within a Web container of a Web server
- J2EE/Java EE applications aren't self contained. In order to be executed, they need to be deployed in a container. In other words, the container provides an execution environment on top of the JVM.
- Also, applications rely on several APIs like JPA, EJB, servlet, JMS, JNDI, etc.
- The role of the EE compliant container is to provide a standard implementation of all or some of these APIs. This means you can theoretically run your application on top of any container as long as it relies on standard APIs.
- From a technical perspective, a container is just another Java SE application with a main() method.
- EE applications on the other hand are a collection of services/beans/servlets/etc. The container finds these components and runs them, providing API implementations, monitoring, scalability, reliability and so on.
- manages nonconfigurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the J2EE platform APIs.

5 container types

3 are server-side containers

- **J2EE server**
  - provides J2EE runtime env.

- **Enterprise JavaBeans (EJB) container**
  - Manages the execution of EJB, runs on the J2EE server.

- **Web container**
  - manage servlets and JavaServer Pages.

2 are client-side

- **Application client container**
  - standalone GUls, console, and batch-type programs — the familiar Java applications started with the java command.

- **Applet container**
  - standalone GUls, console, and batch-type programs — the familiar Java applications started with the java command.

What is Enterprise J2EE EJB Server or Container?

- The EJB server provides an environment that supports the execution of applications developed using Enterprise JavaBeans™ (EJB) components. It manages and coordinates the allocation of resources to the applications. Enterprise beans typically contain the business logic for a J2EE application.
The EJB server must provide one or more EJB containers. An EJB container manages the enterprise beans contained within it. For each enterprise bean, the container is responsible for registering the object, providing a remote interface for the object, creating and destroying object instances, checking security for the object, managing the active state for the object, and coordinating distributed transactions. Optionally, the container can also manage all persistent data within the object.

Enterprise JavaBeans technology supports both transient and persistent objects.

- A transient object is called a session bean.
- A persistent object is called an entity bean.

A session bean exists only for the duration of a single client/server session. A session bean performs operations such as accessing a database or performing calculations. Session beans can be transactional, but normally are not recoverable following a system crash. Session beans can be stateless, or they can maintain conversational state across methods and transactions. A session bean must manage its own persistent data.

An entity bean is an object representation of persistent data maintained in a permanent data store, such as a database. An entity object can manage its own persistence, or it can delegate its persistence to its container.

3. **What is Enterprise Java Beans (EJB) in J2EE ? What is advantage of using EJB ?**

- Enterprise Java beans are J2EE component, which is managed by Container, provided by Application server and encapsulate business logic and forms back end of Enterprise Java application. Main advantage of using EJB is to leverage core services provided by EJB container e.g. transaction management, persistence, security etc. In addition, business logic encapsulate in EJB is good multi-tier design and can be used with different clients e.g. Web, Desktop and mobile. Current version of EJB is EJB 3.0, which is based on annotation.

4. **Can we run EJB in web server like Tomcat ?**

- One of the tricky interview questions on EJB. No you cannot run EJB in web server like Tomcat. You need application server like Glassfish, WebLogic or WebSphere to run Enterprise Java beans. One of the follow-up questions of this is What difference between Application server and web server is.

5. **What were shortcomings of EJB 2.0 ?**

- If you have mentioned in your resume that you have worked on EJB 2.0, then better be prepared for this EJB interview question. It's not difficult to answer if you have indeed worked in EJB 2.0. There are several pains e.g. writing those local and remote interfaces, too many boilerplates coding etc.

6. **What is advantage of putting business logic in EJB over stored procedure?**

- This is another tough Java question on EJB interview. Putting complex business logic in stored procedure is not a good idea but it's been there for long time. Main advantage of using EJB over stored procedure is that you don't need to port your SQL Stored procedure code when you change database e.g. form Sybase to Oracle or Oracle to SQL Server.

7. **What are the services provided by EJB container ?**

- Container is one of the main reasons why Programmer should use Enterprise Java beans. EJB container provides several useful services like :
  - life cycle management of Enterprise Java beans (EJB).
  - Container managed transaction
  - Container managed persistent
  - Security etc.

8. **What is two phase commit? How do you manage distributed transaction in EJB**

- This is one of the trickiest Java questions I have seen in EJB interviews. Two phase commit involves one Transaction manager which controls whole commit process. If you have not used two phase commit or distributed transaction along with EJB then you need to explore this topic in detail to answer this EJB interview question.
Top EJB 3.0 Interview Questions

9. **What is main difference between EJB 2.0 and EJB 3.0?**
   - Simple answer is use of Java annotation, ease of development etc.

10. **Can you name few annotation used in EJB 3.0 and what does he do?**
    - This is another simple EJB 3.0 interview question, which can be easily answered if you have coded few EJB 3.0 beans. Some annotations used are @EJB, @PreDestroy, @PostConstruct etc.

11. **How to call an EJB 3.0 bean from Servlet or JSP?**
    - Use @EJB annotation

EJB Beans and Managed CDI Beans  Contexts and Dependency Injection

12. **Explain the Spring Bean-LifeCycle.**
    - Spring framework is based on IOC (*inversion of control*) so we call it as IOC container also. So Spring beans reside inside the IOC container. **Spring beans are nothing but Plain old java object (POJO).**
    - Following steps explain their life cycle inside the container.
    1. The container will look the bean definition inside configuration file (e.g. bean.xml).
    2. Using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.
    3. If the bean implements the BeanNameAware INTERFACE, the factory calls setBeanName() passing the bean's ID.
    4. If the bean implements the BeanFactoryAware INTERFACE, the factory calls setBeanFactory(), passing an instance of itself.
    5. If there are any BeanPostProcessors associated with the bean, their postProcessBeforeInitialization() methods will be called before the properties for the Bean are set.
    6. If an init() method is specified for the bean, it will be called.
    7. If the Bean class implements the DisposableBean INTERFACE, then the method destroy() will be called when the Application no longer needs the bean reference.
    8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.

To learn more about spring beans and their life cycle I also suggest you take a look at these Spring framework books.

13. **what is Bean Factory, have you used XMLBeanFactory?**
    - Ans: BeanFactory is factory Pattern which is based on IOC (*inversion of control*), design principles. It is used to make a clear separation between application configuration and dependency from actual code.
    - The XmlBeanFactory is one of the implementations of bean Factory which we have used in our project. The org.springframework.beans.factory.xml.XmlBeanFactory is used to create bean instance defined in our XML file.

    ```java
    BeanFactory factory = new XmlBeanFactory(new FileInputStream("beans.xml"));
    Or
    ClassPathResource resource = new ClassPathResource("beans.xml");
    XmlBeanFactory factory = new XmlBeanFactory(resource);
    ```

14. **What are the difference between BeanFactory and ApplicationContext in spring?**
• ApplicationContext is the preferred way of using spring because of functionality provided by it

<table>
<thead>
<tr>
<th>ApplicationContext</th>
<th>BeanFactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here we can have more than one config files possible</td>
<td>In this only one config file or .xml file</td>
</tr>
<tr>
<td>Application contexts can publish events to beans that are registered as listeners</td>
<td>Doesn’t support.</td>
</tr>
<tr>
<td>Support internationalization (I18N) messages</td>
<td>It’s not</td>
</tr>
<tr>
<td>Support application life-cycle events, and validation</td>
<td>Doesn’t support.</td>
</tr>
<tr>
<td>Supports many enterprise services such JNDI access, EJB integration, remoting</td>
<td>Doesn’t support.</td>
</tr>
</tbody>
</table>

15. **What Is an Enterprise Bean?**

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The **business logic** is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, clients can access the inventory services provided by the application.

16. **Benefits of Enterprise Beans**

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services, such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. Provided that they use the standard APIs, these applications can run on any compliant Java EE server.

17. **When to Use Enterprise Beans**

You should consider using enterprise beans if your application has any of the following requirements.

- The application must be scalable. To accommodate a growing number of users, you may need to distribute an application’s components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

18. **Types of Enterprise Beans**

*Table 22-1* summarizes the two types of enterprise beans. The following sections discuss each type in more detail.

| Table 22-1 Enterprise Bean Types |
|--------------------------|-----------------------------|
| **Enterprise Bean Type** | **Purpose**                 |
| Session                  | Performs a task for a client; optionally, may implement a web service |
| Message-driven           | Acts as a listener for a particular messaging type, such as the Java Message Service API |

19. **What Is a Session Bean?**
A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. To access an application that is deployed on the server, the client invokes the session bean’s methods. The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

A session bean is not persistent. (That is, its data is not saved to a database.)

For code samples, see [Chapter 24, Running the Enterprise Bean Examples](https://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html).

### 20. **Types of Session Beans**

Session beans are of three types: stateful, stateless, and singleton.

**Stateful Session Beans**

The state of an object consists of the values of its instance variables. In a **stateful session bean**, the instance variables represent the state of a unique client/bean session. Because the client interacts (“talks”) with its bean, this state is often called the **conversational state**.

As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. When the client terminates, its session bean appears to terminate and is no longer associated with the client.

The state is retained for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

**Stateless Session Beans**

A **stateless session bean** does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply across all clients.

Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

A stateless session bean can implement a web service, but a stateful session bean cannot.

**Singleton Session Beans**

A **singleton session bean** is instantiated once per application and exists for the lifecycle of the application. Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.

Singleton session beans offer similar functionality to stateless session beans but 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. 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.

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 may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

### 21. **When to Use Session Beans**

Stateful session beans are appropriate if any of the following conditions are true.

- The bean’s state represents the interaction between the bean and a specific client.
- The bean needs to hold information about the client across method invocations.
- The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
Behind the scenes, the bean manages the work flow of several enterprise beans. To improve performance, you might choose a stateless session bean if it has any of these traits.

- The bean’s state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send an email that confirms an online order.
- The bean implements a web service.

Singleton session beans are appropriate in the following circumstances.

- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.

22. What Is a Message-Driven Bean?

A message-driven bean is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

23. What Makes Message-Driven Beans Different from Session Beans?

The most visible difference between message-driven beans and session beans is that clients do not access message-driven beans through interfaces. Interfaces are described in the section Accessing Enterprise Beans. Unlike a session bean, a message-driven bean has only a bean class.

In several respects, a message-driven bean resembles a stateless session bean.

- A message-driven bean’s instances retain no data or conversational state for a specific client.
- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.
- A single message-driven bean can process messages from multiple clients.

The instance variables of the message-driven bean instance can contain some state across the handling of client messages, such as a JMS API connection, an open database connection, or an object reference to an enterprise bean object.

Client components do not locate message-driven beans and invoke methods directly on them. Instead, a client accesses a message-driven bean through, for example, JMS by sending messages to the message destination for which the message-driven bean class is the MessageListener. You assign a message-driven bean’s destination during deployment by using GlassFish Server resources.

24. Message-driven beans have the following characteristics.

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

When a message arrives, the container calls the message-driven bean’s onMessage method to process the message. The onMessage method normally casts the message to one of the five JMS message types and handles it in accordance with the application’s business logic. The onMessage method can call helper methods or can invoke a session bean to process the information in the message or to store it in a database.
A message can be delivered to a message-driven bean within a transaction context, so all operations within the `onMessage` method are part of a single transaction. If message processing is rolled back, the message will be redelivered. For more information, see Chapter 25, A Message-Driven Bean Example and Chapter 44, Transactions.

25. **When to Use Message-Driven Beans**

Session beans allow you to send JMS messages and to receive them synchronously but not asynchronously. To avoid tying up server resources, do not use blocking synchronous receives in a server-side component; in general, JMS messages should not be sent or received synchronously. To receive messages asynchronously, use a message-driven bean.

26. **The Contents of an Enterprise Bean**

To develop an enterprise bean, you must provide the following files:

- **Enterprise bean class**: Implements the business methods of the enterprise bean and any lifecycle callback methods.
- **Business interfaces**: Define the business methods implemented by the enterprise bean class. A business interface is not required if the enterprise bean exposes a local, no-interface view.
- **Helper classes**: Other classes needed by the enterprise bean class, such as exception and utility classes.

Package the programming artifacts in the preceding list either into an EJB JAR file (a stand-alone module that stores the enterprise bean) or within a web application archive (WAR) module.

27. **Packaging Enterprise Beans in EJB JAR Modules**

An EJB JAR file is portable and can be used for various applications.

To assemble a Java EE application, package one or more modules, such as EJB JAR files, into an EAR file, the archive file that holds the application. When deploying the EAR file that contains the enterprise bean’s EJB JAR file, you also deploy the enterprise bean to the GlassFish Server. You can also deploy an EJB JAR that is not contained in an EAR file. Figure 22-2 shows the contents of an EJB JAR file.

![Figure 22-2 Structure of an Enterprise Bean JAR](image)

28. **Packaging Enterprise Beans in WAR Modules**

Enterprise beans often provide the business logic of a web application. In these cases, packaging the enterprise bean within the web application’s WAR module simplifies deployment and application organization. Enterprise beans may be packaged within a WAR module as Java programming language class files or within a JAR file that is bundled within the WAR module.

To include enterprise bean class files in a WAR module, the class files should be in the `WEB-INF/classes` directory.
To include a JAR file that contains enterprise beans in a WAR module, add the JAR to the WEB-INF/lib directory of the WAR module.

WAR modules that contain enterprise beans do not require an ejb-jar.xml deployment descriptor. If the application uses ejb-jar.xml, it must be located in the WAR module's WEB-INF directory.

JAR files that contain enterprise bean classes packaged within a WAR module are not considered EJB JAR files, even if the bundled JAR file conforms to the format of an EJB JAR file. The enterprise beans contained within the JAR file are semantically equivalent to enterprise beans located in the WAR module's WEB-INF/classes directory, and the environment namespace of all the enterprise beans are scoped to the WAR module.

For example, suppose that a web application consists of a shopping cart enterprise bean, a credit card processing enterprise bean, and a Java servlet front end. The shopping cart bean exposes a local, no-interface view and is defined as follows:

```java
package com.example.cart;
@Stateless
public class CartBean { ... }
```

The credit card processing bean is packaged within its own JAR file, cc.jar, exposes a local, no-interface view, and is defined as follows:

```java
package com.example.cc;
@Stateless
public class CreditCardBean { ... }
```

The servlet, com.example.web.StoreServlet, handles the web front end and uses both CartBean and CreditCardBean. The WAR module layout for this application looks as follows:

```
WEB-INF/classes/com/example/cart/CartBean.class
WEB-INF/classes/com/example/web/StoreServlet
WEB-INF/lib/cc.jar
WEB-INF/ejb-jar.xml
WEB-INF/web.xml
```