EJB Interview Questions

1. What is EJB?

EJB stands for Enterprise JavaBeans and is widely-adopted server side component architecture for J2EE. It enables rapid development of ission-critical application that are versatile, reusable and portable across middleware while protecting IT investment and preventing vendor lock-in.

2. What is session Facade?

Session Facade is a design pattern to access the Entity bean through local interface than accessing directly. It increases the performance over the network. In this case we call session bean which on turn call entity bean.

3. What is EJB role in J2EE?

EJB technology is the core of J2EE. It enables developers to write reusable and portable server-side business logic for the J2EE platform.

4. What is the difference between EJB and Java beans?

EJB is a specification for J2EE server, not a product; Java beans may be a graphical component in IDE.

5. What are the key features of the EJB technology?

1. EJB components are server-side components written entirely in the Java programming language
2. EJB components contain business logic only - no system-level programming & services, such as transactions, security, life-cycle, threading, persistence, etc. are automatically managed for the EJB component by the EJB server.
3. EJB architecture is inherently transactional, distributed, portable multi-tier, scalable and secure.
4. EJB components are fully portable across any EJB server and any OS.
5. EJB architecture is wire-protocol neutral--any protocol can be utilized like IIOP, JRMP, HTTP, DCOM, etc.

6. What are the key benefits of the EJB technology?

1. Rapid application development
2. Broad industry adoption
3. Application portability
4. Protection of IT investment

7. How many enterprise beans?

There are three kinds of enterprise beans:

1. session beans,
2. entity beans, and
3. message-driven beans.

8. What is message-driven bean?

A message-driven bean combines features of a session bean and a Java Message Service (JMS) message listener, allowing a business component to receive JMS. A message-driven bean enables asynchronous clients to access the business logic in the EJB tier.

9. What are Entity Bean and Session Bean?

Entity Bean is a Java class which implements an Enterprise Bean interface and provides the implementation of the business methods. There are two types: Container Managed Persistence (CMP) and Bean-Managed Persistence (BMP).
Session Bean is used to represent a workflow on behalf of a client. There are two types: Stateless and Stateful. Stateless bean is the simplest bean. It doesn't maintain any conversational state with clients between method invocations. Stateful bean maintains state between invocations.

10. How EJB Invocation happens?

Retrieve Home Object reference from Naming Service via JNDI. Return Home Object reference to the client. Create me a new EJB Object through Home Object interface. Create EJB Object from the Ejb Object. Return EJB Object reference to the client. Invoke business method using EJB Object reference. Delegate request to Bean (Enterprise Bean).

11. Is it possible to share an HttpSession between a JSP and EJB? What happens when I change a value in the HttpSession from inside an EJB?

You can pass the HttpSession as parameter to an EJB method, only if all objects in session are serializable.This has to be considering as passed-by-value that means that it’s read-only in the EJB. If anything is altered from inside the EJB, it won’t be reflected back to the HttpSession of the Servlet Container. The pass-by-reference can be used between EJBs Remote Interfaces, as they are remote references. While it is possible to pass an HttpSession as a parameter to an EJB object, it is considered to be bad practice in terms of object-oriented design. This is because you are creating an unnecessary coupling between back-end objects (EJBs) and front-end objects (HttpSession). Create a higher-level of abstraction for your EJBs API. Rather than passing the whole, fat, HttpSession (which carries with it a bunch of http semantics), create a class that acts as a value object (or structure) that holds all the data you need to pass back and forth between front-end/back-end. Consider the case where your EJB needs to support a non HTTP-based client. This higher level of abstraction will be flexible enough to support it.

12. The EJB container implements the EJBHome and EJBObject classes. For every request from a unique client, does the container create a separate instance of the generated EJBHome and EJBObject classes?

The EJB container maintains an instance pool. The container uses these instances for the EJB Home reference irrespective of the client request. While referring the EJB Object classes the container creates a separate instance for each client request. The instance pool maintenance is up to the implementation of the container. If the container provides one, it is available otherwise it is not mandatory for the provider to implement it. Having said that, yes most of the container providers implement the pooling functionality to increase the performance of the application server. The way it is implemented is, again, up to the implementer.

13. Can the primary key in the entity bean be a Java primitive type such as int?

The primary key can't be a primitive type. Use the primitive wrapper classes, instead. For example, you can use java.lang.Integer as the primary key class, but not int (it has to be a class, not a primitive).

14. Can you control when passivation occurs?

The developer, according to the specification, cannot directly control when passivation occurs. Although for Stateful Session Beans, the container cannot passivate an instance that is inside a transaction. So using transactions can be a strategy to control passivation. The ejbPassivate() method is called during passivation, so the developer has control over what to do during this exercise and can implement the require optimized logic. Some EJB containers, such as BEA Weblogic, provide the ability to tune the container to minimize passivation calls. Taken from the Weblogic 6.0 DTD -The passivation-strategy can be either default or transaction. With the default setting the container will attempt to keep a working set of beans in the cache. With the transaction setting, the container will passivate the bean after every transaction (or method call for a non-transactional invocation).

15. What is the advantage of using Entity bean for database operations, over directly using JDBC API to do database operations? When would I use one over the other?

Entity Beans actually represents the data in a database. It is not that Entity Beans replaces JDBC API. There are two types of Entity Beans Container Managed and Bean Managed. In Container Managed Entity Bean - Whenever the instance of the bean is created the container automatically retrieves the data from the DB/Persistence storage and assigns to the object variables in bean for user to manipulate or use them. For this the developer needs to map the fields in the database to the variables in deployment descriptor files (which varies for each vendor). In the Bean Managed Entity Bean - The developer has to specifically make connection, retrieve values, assign them to the objects in the ejbLoad()

which will be called by the container when it instantiates a bean object. Similarly in the ejbStore() the container saves the object values back the persistence storage. ejbLoad and ejbStore are callback methods and can be only invoked by the container. Apart from this, when you use Entity beans you don't need to worry about database transaction handling, database connection pooling etc. which are taken care by the ejb container.

16. What is EJB QL?

EJB QL is a Query Language provided for navigation across a network of enterprise beans and dependent objects defined by means of container managed persistence. EJB QL is introduced in the EJB 2.0 specification. The EJB QL query language defines finder methods for entity beans with container managed persistence and is portable across containers and persistence managers. EJB QL is used for queries of two types of finder methods: Finder methods that are defined in the home interface of an entity bean and which return entity objects. Select methods, which are not exposed to the client, but which are used by the Bean Provider to select persistent values that are maintained by the Persistence Manager or to select entity objects that are related to the entity bean on which the query is defined.

17. Brief description about local interfaces?

EEJB was originally designed around remote invocation using the Java Remote Method Invocation (RMI) mechanism, and later extended to support to standard CORBA transport for these calls using RMI/IIOP. This design allowed for maximum flexibility in developing applications without consideration for the deployment scenario, and was a strong feature in support of a goal of component reuse in J2EE. Many developers are using EJBs locally, that is, some or all of their EJB calls are between beans in a single container. With this feedback in mind, the EJB 2.0 expert group has created a local interface mechanism. The local interface may be defined for a bean during development, to allow streamlined calls to the bean if a caller is in the same container. This does not involve the overhead involved with RMI like marshalling etc. This facility will thus improve the performance of applications in which co-location is planned. Local interfaces also provide the foundation for container-managed relationships among entity beans with container-managed persistence.

18. What are the special design cares that must be taken when you work with local interfaces?

It is important to understand that the calling semantics of local interfaces are different from those of remote interfaces. For example, remote interfaces pass parameters using call-by-value semantics, while local interfaces use call-by-reference. This means that in order to use local interfaces safely, application developers need to carefully consider potential deployment scenarios up front, then decide which interfaces can be local and which remote, and finally, develop the application code with these choices in mind. While EJB 2.0 local interfaces are extremely useful in some situations, the long-term costs of these choices, especially when changing requirements and component reuse are taken into account, need to be factored into the design decision.

19. What happens if remove( ) is never invoked on a session bean?

In case of a stateless session bean it may not matter if we call or not as in both cases nothing is done. The number of beans in cache is managed by the container. In case of Stateful session bean, the bean may be kept in cache till either the session times out, in which case the bean is removed or when there is a requirement for memory in which case the data is cached and the bean is sent to free pool.

20. What is the difference between Message Driven Beans and Stateless Session beans?

In several ways, the dynamic creation and allocation of message-driven bean instances mimics the behavior of stateless session EJB instances, which exist only for the duration of a particular method call. However, message-driven beans are different from stateless session EJBs (and other types of EJBs) in several significant ways: Message-driven beans process multiple JMS messages asynchronously, rather than processing a serialized sequence of method calls. Message-driven beans have no home or remote interface, and therefore cannot be directly accessed by internal or external clients. Clients interact with message-driven beans only indirectly, by sending a message to a JMS Queue or Topic. Only the container directly interacts with a message-driven bean by creating bean instances and passing JMS messages to those instances as necessary. The Container maintains the entire lifecycle of a message-driven bean; instances cannot be created or removed as a result of client requests or other API calls.

21. How can I call one EJB from inside of another EJB?

EJBs can be clients of other EJBs. It just works. Use JNDI to locate the Home Interface of the other bean, then acquire an instance reference, and so forth.

22. What is an EJB Context?

EJBContext is an interface that is implemented by the container, and it is also a part of the bean-container contract. Entity beans use a subclass of EJBContext called EntityContext. Session beans use a subclass called SessionContext. These EJBContext objects provide the bean class with information about its container, the client using the bean and the bean itself. They also provide other functions. See the API docs and the spec for more details.

23. Is it possible for an EJB client to marshal an object of class java.lang.Class to an EJB?

Technically yes, spec. compliant NO! - The enterprise bean must not attempt to query a class to obtain information about the declared members that are not otherwise accessible to the enterprise bean because of the security rules of the Java language.

24. Is it legal to have static initializer blocks in EJB?

Although technically it is legal, static initializer blocks are used to execute some piece of code before executing any constructor or method while instantiating a class. Static initializer blocks are also typically used to initialize static fields - which may be illegal in EJB if they are read/write - In EJB this can be achieved by including the code in either the ejbCreate(), setSessionContext() or setEntityContext() methods.

25. Is it possible to stop the execution of a method before completion in a SessionBean?

Stopping the execution of a method inside a Session Bean is not possible without writing code inside the Session Bean. This is because you are not allowed to access Threads inside an EJB.

26. What is the default transaction attribute for an EJB?

There is no default transaction attribute for an EJB. Section 11.5 of EJB v1.1 spec says that the deployer must specify a value for the transaction attribute for those methods having container managed transaction. In Weblogic, the default transaction attribute for EJB is SUPPORTS.

27. What is the difference between session and entity beans? When should I use one or the other?

An entity bean represents persistent global data from the database; a session bean represents transient user-specific data that will die when the user disconnects (ends his session). Generally, the session beans implement business methods (e.g. Bank.transferFunds) that call entity beans (e.g. Account.deposit, Account.withdraw)

28. Is there any default cache management system with Entity beans?

In other words whether a cache of the data in database will be maintained in EJB? - Caching data from a database inside the Application Server are what Entity EJBs are used for. The ejbLoad() and ejbStore() methods are used to synchronize the Entity Bean state with the persistent storage(database). Transactions also play an important role in this scenario. If data is removed from the database, via an external application - your Entity Bean can still be alive the EJB container. When the transaction commits, ejbStore() is called and the row will not be found, and the transaction rolled back.

29. Why is ejbFindByPrimaryKey mandatory?

An Entity Bean represents persistent data that is stored outside of the EJB Container/ Server. The ejbFindByPrimaryKey is a method used to locate and load an Entity Bean into the container, similar to a SELECT statement in SQL. By making this method mandatory, the client programmer can be assured that if they have the primary key of the Entity Bean,

then they can retrieve the bean without having to create a new bean each time - which would mean creating duplications of persistent data and break the integrity of EJB.

30. Why do we have a remove method in both EJBHome and EJBObject?

With the EJBHome version of the remove, you are able to delete an entity bean without first instantiating it (you can provide a PrimaryKey object as a parameter to the remove method). The home version only works for entity beans. On the other hand, the Remote interface version works on an entity bean that you have already instantiated. In addition, the remote version also works on session beans (stateless and Stateful) to inform the container of your loss of interest in this bean.

31. How can I call one EJB from inside of another EJB?

EJBs can be clients of other EJBs. It just works. Use JNDI to locate the Home Interface of the other bean, then acquire an instance reference, and so forth.

32. What is the difference between a Server, a Container, and a Connector?

An EJB server is an application, usually a product such as BEA Weblogic, that provides (or should provide) for concurrent client connections and manages system resources such as threads, processes, memory, database connections, network connections, etc. An EJB container runs inside (or within) an EJB server, and provides deployed EJB beans with transaction and security management, etc. The EJB container insulates an EJB bean from the specifics of an underlying EJB server by providing a simple, standard API between the EJB bean and its container. A Connector provides the ability for any Enterprise Information System (EIS) to plug into any EJB server which supports the Connector architecture. See Sun’s J2EE Connectors for more in-depth information on Connectors.

33. How is persistence implemented in enterprise beans?

Persistence in EJB is taken care of in two ways, depending on how you implement your beans: container managed persistence (CMP) or bean managed persistence (BMP) For CMP, the EJB container which your beans run under takes care of the persistence of the fields you have declared to be persisted with the database - this declaration is in the deployment descriptor. So, anytime you modify a field in a CMP bean, as soon as the method you have executed is finished, the new data is persisted to the database by the container. For BMP, the EJB bean developer is responsible for defining the persistence routines in the proper places in the bean, for instance, the ejbCreate(), ejbStore(), ejbRemove() methods would be developed by the bean developer to make calls to the database. The container is responsible, in BMP, to call the appropriate method on the bean. So, if the bean is being looked up, when the create() method is called on the Home interface, then the container is responsible for calling the ejbCreate() method in the bean, which should have functionality inside for going to the database and looking up the data.

34. What is an EJB Context?

EJBContext is an interface that is implemented by the container, and it is also a part of the bean-container contract. Entity beans use a subclass of EJBContext called EntityContext. Session beans use a subclass called SessionContext. These EJBContext objects provide the bean class with information about its container, the client using the bean and the bean itself. They also provide other functions. See the API docs and the spec for more details.

35. Is method overloading allowed in EJB?

Yes you can overload methods should synchronization primitives be used on bean methods? - No. The EJB specification specifically states that the enterprise bean is not allowed to use thread primitives. The container is responsible for managing concurrent access to beans at runtime.

36. Are we allowed to change the transaction isolation property in middle of a transaction?

No. You cannot change the transaction isolation level in the middle of transaction.

37. For Entity Beans, What happens to an instance field not mapped to any persistent storage, when the bean is passivated?

The specification infers that the container never serializes an instance of an Entity bean (unlike Stateful session beans). Thus passivation simply involves moving the bean from the ready to the pooled bin. So what happens to the contents of an instance variable is controlled by the programmer. Remember that when an entity bean is passivated the instance gets logically disassociated from its remote object. Be careful here, as the functionality of passivation/activation for Stateless Session, Stateful Session and Entity beans is completely different. For entity beans the ejbPassivate method notifies the entity bean that it is being disassociated with a particular entity prior to reuse or for dereference.

38. What is a Message Driven Bean, what functions does a message driven bean have and how do they work in collaboration with JMS?

Message driven beans are the latest addition to the family of component bean types defined by the EJB specification. The original bean types include session beans, which contain business logic and maintain a state associated with client sessions, and entity beans, which map objects to persistent data. Message driven beans will provide asynchrony to EJB based applications by acting as JMS message consumers. A message bean is associated with a JMS topic or queue and receives JMS messages sent by EJB clients or other beans. Unlike entity beans and session beans, message beans do not have home or remote interfaces. Instead, message driven beans are instantiated by the container as required. Like stateless session beans, message beans maintain no client-specific state, allowing the container to optimally manage a pool of message-bean instances. Clients send JMS messages to message beans in

exactly the same manner as they would send messages to any other JMS destination. This similarity is a fundamental design goal of the JMS capabilities of the new specification. To receive JMS messages, message driven beans implement the javax.jms.MessageListener interface, which defines a single onMessage() method. When a message arrives, the container ensures that a message bean corresponding to the message topic/queue exists (instantiating it if necessary), and calls its onMessage method passing the client's message as the single argument. The message bean's implementation of this method contains the business logic required to process the message. Note that session beans and entity beans are not allowed to function as message beans.

39. Does RMI-IIOP support code downloading for Java objects sent by value across an IIOP connection in the same way as RMI does across a JRMP connection?

Yes. The JDK 1.2 supports the dynamic class loading. The EJB container implements the EJBHome and EJBObject classes. For every request from a unique client,

40. Does the container create a separate instance of the generated EJBHome and EJBObject classes?

The EJB container maintains an instance pool. The container uses these instances for the EJB Home reference irrespective of the client request. While referring the EJB Object classes the container creates a separate instance for each client request. The instance pool maintenance is up to the implementation of the container. If the container provides one, it is available otherwise it is not mandatory for the provider to implement it. Having said that, yes most of the container providers implement the pooling functionality to increase the performance of the application server. The way it is implemented is again up to the implementer.

41. What is the advantage of putting an Entity Bean instance from the Ready State to Pooled State?

The idea of the Pooled State is to allow a container to maintain a pool of entity beans that has been created, but has not been yet synchronized or assigned to an EJBObject. This mean that the instances do represent entity beans, but they can be used only for serving Home methods (create or findBy), since those methods do not relay on the specific values of the bean. All these instances are, in fact, exactly the same, so, they do not have meaningful state. Jon Thorarinsson has also added: It can be looked at it this way: If no client is using an entity bean of a particular type there is no need for cachig it (the data is persisted in the database). Therefore, in such cases, the container will, after some time, move the entity bean from the Ready State to the Pooled state to save memory. Then, to save additional memory, the container may begin moving entity beans from the Pooled State to the Does Not Exist State, because even though the bean's cache has been cleared, the bean still takes up some memory just being in the Pooled State.

42. What is the need of Remote and Home interface. Why can’t it be in one?

The main reason is because there is a clear division of roles and responsibilities between the two interfaces. The home interface is your way to communicate with the container, that is that is responsible of creating, locating even removing one or more beans. The remote interface is your link to the bean that will allow you to remotely access to all its methods and members. As you can see there are two distinct elements (the container and the beans) and you need two different interfaces for accessing to both of them.

43. Can I develop an Entity Bean without implementing the create() method in the home interface?

As per the specifications, there can be 'ZERO' or 'MORE' create() methods defined in an Entity Bean. In cases where create() method is not provided, the only way to access the bean is by knowing its primary key, and by acquiring a handle to it by using its corresponding finder method. In those cases, you can create an instance of a bean based on the data present in the table. All one needs to know is the primary key of that table. I.e. a set a columns that uniquely identify a single row in that table. Once this is known, one can use the 'getPrimaryKey()' to get a remote reference to that bean, which can further be used to invoke business methods.
What is the difference between Context, InitialContext and Session Context? How they are used? javax.naming.Context is an interface that provides methods for binding a name to an object. It's much like the RMI Naming.bind() method.
javax.naming.InitialContext is a Context and provides implementation for methods available in the Context interface.
Where as SessionContext is an EJBContext objects that is provided by the EJB container to a SessionBean in order for the SessionBean to access the information and/or services or the container.
There is EntityContext too which is also and EJBContext object that'll be provided to an EntityBean for the purpose of the EntityBean accessing the container details. In general, the EJBContext (SessionContext and EntityContext), AppletContext and ServletContext help the corresponding Java objects in knowing about its 'context' [environment in which they run], and to access particular information and/or service. Whereas, the javax.naming.Context is for the purpose of 'NAMING' [by the way of referring to] an object.

44. Why an onMessage call in Message-driven bean is always a separate transaction?

EJB 2.0 specification: "An onMessage call is always a separate transaction, because there is never a transaction in progress when the method is called." When a message arrives, it is passed to the Message Driven Bean through the onMessage() method, that is where the business logic goes. Since there is no guarantee when the method is called and when the message will be processed, is the container that is responsible of managing the environment, including transactions.

45. Why are ejbActivate() and ejbPassivate() included for stateless session bean even though they are never required as it is a no conversational bean?

To have a consistent interface, so that there is no different interface that you need to implement for Stateful Session Bean and Stateless Session Bean. Both Stateless and Stateful Session Bean implement javax.ejb.SessionBean and this would not be possible if stateless session bean is to remove ejbActivate and ejbPassivate from the interface.

46. Static variables in EJB should not be relied upon as they may break in clusters. Why?

Static variables are only ok if they are final. If they are not final, they will break the cluster. What that means is that if you cluster your application server (spread it across several machines) each part of the cluster will run in its own JVM.
Say a method on the EJB is invoked on cluster 1 (we will have two clusters - 1 and 2) that causes value of the static variable to be increased to 101. On the subsequent call to the same EJB from the same client, a cluster 2 may be invoked to handle the request. A value of the static variable in cluster 2 is still 100 because it was not increased yet and therefore your application ceases to be consistent. Therefore, static non-final variables are strongly discouraged in EJBs.

47. If I throw a custom ApplicationException from a business method in Entity bean which is participating in a transaction, would the transaction be rolled back by container?

EJB Transaction is automatically rolled back only when a SystemException (or a subtype of it) is thrown. Your ApplicationException can extend from javax.ejb.EJBException, which is a sub class of RuntimeException. When an EJBException is encountered the container rolls back the transaction. EJB Specification does not mention anything about Application exceptions being sub-classes of EJBException. You can tell container to rollback the transaction, by using setRollBackOnly on SessionContext/EJBContext object as per type of bean you are using.

48. Does Stateful Session bean support instance pooling?

Stateful Session Bean conceptually doesn't have instance pooling.

49. Can I map more than one table in a CMP?

No, you cannot map more than one table to a single CMP Entity Bean. CMP has been, in fact, designed to map a single table.

50. Can I invoke Runtime.gc() in an EJB?

You shouldn't. What will happen depends on the implementation, but the call will most likely be ignored.

51. Can a Session Bean be defined without ejbCreate() method?

The ejbCreate() methods is part of the bean's lifecycle, so, the compiler will not return an error because there is no ejbCreate() method.
However, the J2EE spec is explicit:

- the home interface of a Stateless Session Bean must have a single create() method with no arguments, while the session bean class must contain exactly one ejbCreate() method, also without arguments.
- Stateful Session Beans can have arguments (more than one create method)

52. How to implement an entity bean which the PrimaryKey is an auto numeric field?

The EJB 2 Spec (10.8.3 - Special case: Unknown primary key class) says that in cases where the Primary Keys are generated automatically by the underlying database, the bean provider must declare the findByPrimaryKey method to return java.lang.Object and specify the Primary Key Class as java.lang.Object in the Deployment Descriptor.

53. When defining the Primary Key for the Enterprise Bean, the Deployer using the Container Provider's tools will typically add additional container-managed fields to the concrete subclass of the entity bean class.

In this case, the Container must generate the Primary Key value when the entity bean instance is created (and before ejbPostCreate is invoked on the instance.)

54. What is clustering?

Clustering is grouping machines together to transparently provide enterprise services. Clustering is an essential piece to solving the needs for today's large websites.
The client does not know the difference between approaching one server and approaching a **cluster of servers**.

55. Is it possible to share an HttpSession between a JSP and EJB?

What happens when I change a value in the HttpSession from inside an EJB? You can pass the HttpSession as parameter to an EJB method, **only if all objects in session are serializable.This**. This has to be consider as "passed-by-value", that means that it's read-

only in the EJB. If anything is altered from inside the EJB, it won't be reflected back to the HttpSession of the Servlet Container.

56. If my session bean with single method insert record into 2 entity beans, how can know that the process is done in same transaction (the attributes for these beans are Required)?

If your session bean is using bean-managed transactions, you can ensure that the calls are handled in the same transaction by :

```
        javax.transaction.UserTransaction tran= null;
        try{
        tran=ctx.getUserTransaction();
        tran.begin();
myBeanHome1.create(....);
        myBeanHome2.create(...);
        tran.commit();
        }catch(...){}
```

You may want to check if you're already running in a transaction by calling tran.getStatus().

57. When should I adopt BMP and when I should use CMP?

You can use CMP and BMP beans in the same application... obviously, a bean can be BMP or CMP, not both at the same time (they are mutually exclusive).
There is a common approach that is normally used and considered a good one. You should start developing CMP beans, unless you require some kind of special bean, like multi-tables, that cannot be completely realized with a single bean. Then, when you realize that you need something more or that you would prefer handling the persistence (performance issue are the most common reason), you can change the bean from a CMP to a BMP.

58. What's different in Enterprise JavaBeans 1.1?

The most significant changes are listed below:

- Entity bean support, both container- and bean-managed persistence, is required.
- Java RMI-IIOP argument and reference types must be supported, but any protocol can still be used including IIOP, JRMP, HTTP, or a proprietary protocol. In other words, the client API must support the Java RMI-IIOP programming model for portability, but the underlying protocol can be anything.
- The javax.ejb.depoyment package has been dropped in favor of a XML based deployment descriptor
- Declarative security authorization (access control) has changed to be more role driven. Also the runAs declarations have been eliminated.
- Declarative isolation levels are no longer available. Isolation levels are now managed explicitly through JDBC (BMP), the database or other vendor specific mechanisms.
- The bean-container contract as been enhanced to include a default JNDI context for accessing properties, resources, (JDBC, JMS, etc), and other beans.

- The basic EJB roles have been expanded and redefined to better separate responsibilities involved in the development, deployment, and hosting of enterprise beans.

59. What is Enterprise JavaBeans?

Enterprise JavaBeans (EJB) is Sun Microsystems' specification for a distributed object system similar to CORBA and Microsoft Transaction Server, but based on the Java platform. EJB specifies how developers should build components that can be accessed remotely and how EJB vendors should support those components. EJB components, called enterprise beans, automatically handle transactions, persistence, and authorization security, so that the developer can focus on the business logic.

60. What is an enterprise bean?

An enterprise bean is a server-side component -- defined in the Java technology -- which adheres to the Enterprise JavaBeans server-side component model. A server-side component is business object that can be accessed remotely. Many server-side component models exist: CORBA specifies CORBA objects; Microsoft Transaction Server (MTS) defines COM/DCOM; and EJB specifies enterprise beans.
Enterprise beans can be developed to represent business concepts like Employee, Order, Travel Agent, etc. Enterprise beans can be assembled into applications that solve enterprise business problems.
EJB has two basic types of enterprise beans: Session and Entity. Depending on the type of enterprise bean used, features like persistence, transactions, security, and multiple concurrent access can be managed automatically.

61. Are Enterprise JavaBeans and JavaBeans the same thing?

Enterprise JavaBeans and JavaBeans are not the same thing; nor is one an extension of the other. They are both component models, based on Java, and created by Sun Microsystems, but their purpose and packages (base types and interfaces) are completely different.
JavaBeans
The original JavaBeans specification is based on the java.beans package which is a standard package in the JDK. Components built on the JavaBeans specification are ***intra*process** components that live in one address space and are typically used for Graphical User Interface (GUI) as visual widgets like buttons, tables, HTML viewers, etc.
Enterprise JavaBeans
The EJB specification is based on the javax.ejb package, which is a standard extension package. Components built on the EJB specification are ***intra*process** components that live in multiple address spaces as distributed object. These components are used as transactional business objects that are accessed as remote objects.

62. How does passivation work in stateful session beans?

Unlike entity beans and stateless session beans, stateful session bean are usually evicted from memory when they are passivated. This is not true of all vendors but this view serves as good model for understanding the concepts of passivation in session beans.

When a stateful bean experiences a lull in use -- between client invocations and transactions -- the container may choose to passivate the stateful bean instance. To conserve resources the bean instance is evicted from memory (dereferenced and garbage collected). When the EJB object receives a new client request, a new stateful instance is instantiated and associate with the EJB object to handle the request.
Stateful beans maintain a conversational state, which must be preserved before the bean instance is evicted from memory. To accomplish this, the container will write the conversational state of the bean instance to a secondary storage (usually disk). Only the non-transient serializable instance fields are preserved. When the bean is activated the new instance is populated with the preserved state. References to live resources like the EJBContext, DataSource, JNDI ENC, and other beans must also be maintained somehow -- usually in memory -- by the container.
The javax.ejb.SessionBean interface provides two callback methods that notify the bean instance it is about to passivated or was just activated. The ejbPassivate( ) method notifies the bean instance that it is about have its conversational state written to disk and be evicted from memory. Within this method the bean developer can perform operations just prior to passivation like closing open resources. The ejbActivate( ) method is executed just after a new bean instance has been instantiated and populated with conversational state from disk. The bean developer can use the ejbActivate( ) method to perform operations just prior to servicing client request, like opening resources.

63. How does a client application create a transaction object?

How you gain access to UserTransaction objects varies depending on the type of client. Enterprise JavaBeans provides two types of transaction management:

- Container-managed transactions. As the name implies, the EJB container makes the decisions (based on the deployment descriptor's trans-attribute setting) regarding how to bundle operations into transactions and then works with the transaction manager, which manages the transaction processing.
- Bean-managed transactions. In this case, a session bean obtains the UserTransaction object via the EJBContext using the getUserTransaction() method.

JMS clients can bundle several messages in a transaction simply by using a transactional session--a UserTransaction object is not required. To create a transactional session, use either QueueConnection.createQueueSession() or TopicConnection.createTopicSession() with true as the first argument. (Some JMS implementations support JTA, so that it's also possible to obtain a UserTransaction object from the JMS server.)
In other environments, for example, a web server that supports servlets and/or JavaServer Pages (JSP), a JMS server, and others, you obtain a UserTransaction object via JNDI. Of course, for a servlet to obtain a UserTransaction object, there must be a JTS-capable server to deliver the object.
Typically, the server provides the JNDI look-up name either directly or via a system or server property. For example, with the WebLogic server, you would use a code segment similar to the following:

```
...
Context c = new InitialContext();
UserTransaction ut = (UserTransaction)
c.lookup("javax.jts.UserTransaction");
ut.begin();
// perform multiple operations...
```

```
ut.commit()
...
```

With J2EE implementations, you obtain the UserTransaction object with a code segment similar to the following:

```
...
Context c = new InitialContext();
UserTransaction ut = (UserTransaction)
c.lookup("java:comp/UserTransaction");
ut.begin();
// perform multiple operations...
ut.commit()
...
```

If the environment provides the UserTransaction object via a system property, you would use a code segment similar to the following:

```
...
String transName = System.getProperty("jta.UserTransaction");
Context c = new InitialContext();
UserTransaction ut = (UserTransaction) c.lookup(transName);
ut.begin();
// perform multiple operations...
ut.commit()
...
```

JNDI remote look-up names and property names vary, of course, across servers/ environment.

64. Do JTS implementations support nested transactions?

A JTS transaction manager must support flat transactions; support of nested transactions is optional. If a client begins a transaction, and within that transaction begins another transaction, the latter operation will throw a NotSupportedException if the JTS implementation does not support nested transactions.
Keep in mind that even if the JTS implementation supports nested transactions, this transaction manager-level support does not guarantee support for nested transactions in an application. For example, the EJB 1.1 specification does not support nested transactions.

65. Why would a client application use JTA transactions?

One possible example would be a scenario in which a client needs to employ two (or more) session beans, where each session bean is deployed on a different EJB server and each bean performs operations against external resources (for example, a database) and/or is managing one or more entity beans. In this scenario, the client's logic could required an all-or-nothing guarantee for the operations performed by the session beans; hence, the session bean usage could be bundled together with a JTA UserTransaction object.
In the previous scenario, however, the client application developer should address the question of whether or not it would be better to encapsulate these operations in yet another session bean, and allow the session bean to handle the transactions via the EJB container.

In general, lightweight clients are easier to maintain than heavyweight clients. Also, EJB environments are ideally suited for transaction management.

66. How does a session bean obtain a JTA UserTransaction object?

If it's necessary to engage in explicit transaction management, a session bean can be designed for bean-managed transactions and obtain a UserTransaction object via the EJBContext using the getUserTransaction() method. (It may also use JNDI directly, but it's simpler to use this convenience method.)

67. Why would a session bean use bean-managed transactions?

In some situations, it's necessary for a (stateful) session bean to selectively control which methods participate in transactions, and then take over the bundling of operations that form a logical unit of work.

68. How does an enterprise bean that uses container-managed transactions obtain a JTA UserTransaction object?

It doesn't! By definition, container-managed transaction processing implies that the EJB container is responsible for transaction processing. The session bean has only limited control of transaction handling via the transaction attribute.

69. Is it possible for a stateless session bean to employ a JTA UserTransaction object?

Yes, but with restrictions. By definition, a stateless session bean has no state; hence, each method invocation must be independent. (The bean can be "swapped out" between method invocations.) Thus, a stateless session bean can obtain a UserTransaction object via the EJBContext using the getUserTransaction() method, but it must start and finish each transaction within the scope of a method invocation.

70. How do you configure a session bean for bean-managed transactions?

You must set transaction-type in the deployment descriptor.

71. How does an entity bean obtain a JTA UserTransaction object?

It doesn't. Entity beans do not employ JTA transactions; that is, entity beans always employ declarative, container-managed transaction demarcation. Entity beans, by definition, are

somewhat tightly coupled (via the EJB container and server) to a datastore; hence, the EJB container is in the best position to manage transaction processing.

72. Is it necessary for an entity bean to protect itself against concurrent access from multiple transactions?

No. One of the motivations for using a distributed component architecture such as Enterprise JavaBeans is to free the business logic programmer from the burdens that arise in multiprogramming scenarios.

73. What are the constraints or drawbacks of container managed EJB's?

CMP in beans depends a lot on the EJB vendor implementation and utilities. With some implementations container-managed entity beans can only by mapped to one table, while other implemenations offer Multiple table mappings to a single bean. The bottom line,It depends on the container provider being used.

74. Is entity data persisted at the end of a transaction or at any time?

Depends on what you mean by "persisted". Data that is part of a transaction like database rows are persisted depending on the success of the transaction. If the transaction manager determines that the transaction was successful or there were no problems during any of the steps invoved in it, the data is committed, or otherwise rolled back.
The container, on the other hand, invokes certain state transition lifecycle methods to conserve resources. This involves passivation and activation of the bean or instance swapping. This happens independent of the transaction since the client never interacts directly with the bean instance but with the server's implementation of the EJBObject.

75. How do enterprise beans access native libraries?

In short, they don't.
The EJB 1.1 Specification, section 18.1.2 (Programming Restrictions) lists some things that enterprise beans cannot do. In particular:

> Â· The enterprise bean must not attempt to load a native library.
> *This function is reserved for the EJB Container. Allowing the enterprise bean to l*

76. How do I implement a logging system in my beans?

> **Using java.io is prohibited**
> Using the java.io package from within a bean is prohibited. The EJB 1.1 Specificat
> (Programming Restrictions) states the following:
>
> ï,· An enterprise bean must not use the java.io package to attempt to access files
> directories in the file system.

The file system APIs are not well-suited for business components to access data. B
components should use a resource manager API, such as JDBC API, to store data.

Alternative Solutions

To perform logging operations you must you a resource connection supported by the
EJB servers may support several resource connection options, which can be used to
below:

Â· **JDBC (javax.sql.DataSource) :** Write log events in to a relational database.
Â· **URL (java.net.URL) :** Write log events to a custom logging server or post them t
Â· **JavaMail (javax.mail.Session) :** E-mail log events to a special account
Â· **JMS (javax.jms.QueueConnectionFactory | javax.jms.TopicConnectionFactory) :** Sen

77. What is a container?

Enterprise beans are software components that run in a special environment called an EJB container. The container hosts and manages an enterprise bean in the same manner that a Java WebServer hosts a Servlet or an HTML browser hosts a Java applet. An enterprise bean cannot function outside of an EJB container. The EJB container manages every aspect of an enterprise bean at run time including remote access to the bean, security, persistence, transactions, concurrency, and access to and pooling of resources.
The container isolates the enterprise bean from direct access by client applications. When a client application invokes a remote method on an enterprise bean, the container first intercepts the invocation to ensure persistence, transactions, and security are applied properly to every operation a client performs on the bean. The container manages security, transactions, and persistence automatically for the bean, so the bean developer doesn't have to write this type of logic into the bean code itself. The enterprise bean can focus on encapsulating business rules, while the container takes care of everything else.

Containers will manage many beans simultaneously in the same fashion that a Java WebServer manages many Servlets. To reduce memory consumption and processing, containers pool resources and manage the lifecycles of all the beans very carefully. When a bean is not being used a container will place it in a pool to be reused by another client, or possibly evict it from memory and only bring it back when its needed. Because client applications don't have direct access to the beans -- the container lies between the client and bean -- the client application is completely unaware of the containers resource management activities. A bean that is not in use, for example, might be evicted from memory on the server, while its remote reference on the client remains intact. When the client invokes a method on the remote reference, the container simply re-incarnates the bean to service the request. The client application is unaware of the entire process.
An enterprise bean depends on the container for everything it needs. If an enterprise bean needs to access a JDBC connection or another enterprise bean, it does so through the container; if an enterprise bean needs to access the identity of its caller, obtain a reference to itself, or access properties it does so through the container. The enterprise bean interacts with its container through one of three mechanisms: callback methods, the EJBContext interface, or JNDI.

- **Callback Methods:** Every bean implements a subtype of the EnterpriseBean interface which defines several methods, called callback methods. Each callback method alerts the bean of a different event in its lifecycle and the container will invoke these methods to notify the bean when it's about to pool the bean, persist its state to the database, end a transaction, remove the bean from memory, etc. The callback methods give the bean a chance to do some housework immediately before or after some event. Callback methods are discussed in more detail in other sections.

- **EJBContext:** Every bean obtains an EJBContext object, which is a reference directly to the container. The EJBContext interface provides methods for interacting with the container so that that bean can request information about its environment like the identity of its client, the status of a transaction, or to obtain remote references to itself.
- **JNDI:** Java Naming and Directory Interface is a Java extension API for accessing naming systems like LDAP, NetWare, file systems, etc. Every bean automatically has access to a special naming system called the Environment Naming Context (ENC). The ENC is managed by the container and accessed by beans using JNDI. The JNDI ENC allows a bean to access resources like JDBC connections, other enterprise beans, and properties specific to that bean.

The EJB specification defines a bean-container contract, which includes the mechanisms (callbacks, EJBContext, JNDI ENC) described above as well as a strict set of rules that describe how enterprise beans and their containers will behave at runtime, how security access is checked, transactions are managed, persistence is applied, etc. The bean-container contract is designed to make enterprise beans portable between EJB containers so that enterprise beans can be developed once then run in any EJB container. Vendors like BEA, IBM, and Gemstone sell application servers that include EJB containers. Ideally, any enterprise bean that conforms to the specification should be able to run in any conformant EJB container.

Portability is central to the value that EJB brings to the table. Portability ensures that a bean developed for one container can be migrated to another if another brand offers more performance, features, or savings. Portability also means that the bean developer's skills can be leveraged across several EJB container brands, providing organizations and developers with better opportunities.

In addition to portability, the simplicity of the EJB programming model makes EJB valuable. Because the container takes care of managing complex tasks like security, transactions, persistence, concurrency and resource management the bean developer is free to focus attention on business rules and a very simple programming model. A simple programming model means that beans can be developed faster without requiring a Ph.D. in distributed objects, transactions and other enterprise systems. EJB brings transaction processing and disributed objects development into the mainstream.

78. What makes a Java class an enterprise bean?

An enterprise bean is composed of many parts, not just a single class. Essentially, an enterprise bean is constructed with a bean class, remote interface, home interface and deployment descriptor. These constituents are discussed below.

- ï‚· A **bean class** is the implementation class of the bean that defines its business, persistence, and passivation logic. The bean class implements either the javax.ejb.EntityBean or javax.ejb.SessionBean interface and runs inside the EJB container. Instances of the bean class service client request indirectly; instances of the bean class are not visible to the client.
- ï‚· The **remote interface** defines the business methods that will be visible to the client's that use the enterprise bean. The remote interface extends the javax.ejb.EJBObject interface and is implemented by a remote (distributed object) reference. Client applications interact with the enterprise bean through its remote interface.
- ï‚· The **home interface** defines the create, delete (remove), and query methods for an enterprise bean type. The home interface extends the javax.ejb.EJBHome interface and is implemented by a remote (distributed object) reference. The client application will use the home interface to create beans, find existing beans, and remove specific beans.

- ï‚· The **deployment descriptor** is used to describe the enterprise bean's runtime behavior to the container. Among other things the deployment descriptor allows the transaction, persistence, and authorization security behavior of a bean to be defined using declarative attributes. This greatly simplifies the programming model when developing beans.

An enterprise bean represents the sum of all these parts (remote, home, bean class, and deployment descriptor) as one component. An enterprise bean is not an enterprise bean if any one of these parts is missing. A change to anyone of these parts -- changing even one attribute in the deployment descriptor for example -- creates an entirely new enterprise bean.

79. While deploying CMP entity beans, which fields in the bean are container-managed and how are they identified?

Container-managed fields may be specified in the bean's deployment descriptor. An entity bean, for example, has an XML deployment descriptor containing elements similar to the following:

```
<enterprise-beans>
<entity>
<description>This entity bean models an audio compact disc.</description>
<ejb-name>MusicCDBean</ejb-name>
<home>musicstore.MusicCDHome</home>
<remote>musicstore.MusicCD</remote>
<ejb-class>musicstore.MusicCDBean</ejb-class>
<persistence-type>Container</persistence-type>
<prim-key-class>musicstore.MusicCDPK</prim-key-class>
<reentrant>False</reentrant>

<cmp-field><field-name>upc</field-name></cmp-field>
<cmp-field><field-name>title</field-name></cmp-field>
<cmp-field><field-name>artist</field-name></cmp-field>
<cmp-field><field-name>type</field-name></cmp-field>
<cmp-field><field-name>price</field-name></cmp-field>
</entity>
</enterprise-beans>
```

In the above deployment descriptor, the container-managed fields are specified to be upc, title, artist, type, and price.
While the deployment descriptor provides information about the container-managed fields for use during deployment, the details of how these fields are mapped into the database (or other persistent storage mechanism) are controlled by the container-specific deployment process itself. To learn more about the container-specific deployment process, you will need to consult your container vendor's documentation.

80. Does the EJB programming model support inheritance?

Inheritance is supported in EJB in a limited fashion. Enterprise beans are made up of several parts including: a remote interface; a home interface, a bean class (implementation); and a deployment descriptor

The remote interface, which extends javax.ejb.EJBObject can be a subtype or a super-type of remote interfaces of other beans. This is also true of the home interface, which extends javax.ejb.EJBHome. The bean class, which implements either javax.ejb.EntityBean or javax.ejb.SessionBean can also be a subtype or super-type of the bean class used by another enterprise bean. Deployment descriptors are XML files, so there is no Object-Oriented (OO) inheritance in the deployment descriptor.
Because an enterprise bean is not one object -- its the composition of several parts -- traditional OO inheritance is not possible. The constituent Java parts (remote, home, bean class) of an enterprise bean may themselves subtype or serve as super-type, but the bean as a whole (the sum of its parts) doesn't support inheritance.

81. How should complex find operations be implemented?

In bean-managed persistence (BMP) complex find operations are not difficult to implement, because you have complete control over how a find operation works through the ejbFind methods in the bean class. ejbFind methods in BMP entities can span multiple tables and even different data sources to locate the right entity beans.
With container-managed persistence (CMP) its more difficult because you are dependent on the versatility of the EJB vendor. In other words, if the vendor does not support sophisticated find operations or syntax, its more difficult to declare complex find operations at deployment time. With CMP you have a couple options:

- Â· **Convert the CMP bean to a BMP bean and hand code the ejbFind methods yourself.** This is a classic scenario for using BMP over CMP; when the EJB vendor is not sophisticated enough to support a bean's data access needs.
- Â· **Use a session bean to obtain the data you need.** When a search operation becomes to complex to implement in a single bean its a good indication that the search operation is not appropriate for a find method. Search operations that span the data encapsulated by several different entity beans should be placed in a session bean with the emphasis on returning only the data needed, not necessarily bean references. Data can be returned in tabular format instead of bean references.

NOTE:
A common design error is to implement search operations that filter results of multi-entity find requests implemented by other entity beans. This should be avoided. If you can not find the entity beans in one find request, then you should use a search method in a session bean.

82. Can I use Threads in a enterprise bean?

No. The thread management is done by the container for you. As a bean developer you are not allowed to use threads.
Section 18.1.2 of the EJB 1.1 specification states:

- Â· The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread; or to change a threadâ€™s priority or name. The enter-prise bean must not attempt to manage thread groups.

These functions are reserved for the EJB Container. Allowing the enterprise bean to manage threads would decrease the Containerâ€™s ability to properly manage the runtime environment.

83. Why are beans not allowed to create their own threads?

Enterprise beans exist inside a container at run time. The container is responsible for managing every aspect of the enterprise bean's life including: transactions, access control, persistence, resource pooling, etc. In order for the container to manage the runtime environment of a bean, it must have complete control over the threads that access and run within a bean. This means that beans can not start or manage their own threads. Containers deny enterprise beans the privilege to manage threads for three basic reasons: Resource management, security, and thread-sensitive storage.

Resource Management

Containers manage every aspect of the runtime environment used by enterprise beans including transactions, access control, life cycle, resource connections, VM security, class loading, and threads. This allows the container to conserve as many resources as possible, which is important when there are hundreds of enterprise beans servicing thousands of clients. Without strict management of resources like memory and threads, EJB systems might consume to many resources (memory and cycles), which would result in a slow system, a prospect that is untenable in a high-transaction environment. Threads started and managed by enterprise beans would not be managed by the container, which would make it difficult to conserve resources.

Security

There is no way for a container system to know in advance that a bean's use of threads is benign. While intentions may be sincere it is possible -- probably inevitable -- that developers would create malignant beans that spawn so many threads that the entire system slows down. One bean instance's misuse of threads or the commutative effect of many instances could cause a system slowdown. This is an insurgent denial of service, where the beans themselves sabotage a system's ability to respond to client requests. Security is a very good reason for denying bean's the privilege of starting and managing their own threads.

Thread-Specific Storage

Thread-Specific Storage (TSS) is an established and common technique employed by vendors to propagate and track client requests through the container system. It involves associating data with a thread. The data may be information about the client's identity, the transaction context, and other information, which can be accessed by any part of the container without having to pass the data explicitly. This is especially useful when enterprise beans invoke other enterprise beans or access resources, because it provides a convenient and transparent mechanism for transferring information about the who is making the request and under what circumstances. Each vendor will use the TSS technique differently according to the mechanics of their server. Threads started and managed by the enterprise bean explicitly would not have the proper TSS -- that would require intimate knowledge and access to the vendors container system. Without the right TSS the enterprise bean's threads can not operate within the container system properly. This is another reason why bean are not allowed to start and manage their own threads, it would short-circuit the vendor's use of TSS.

84. How does EJB support polymorphism?

Because an EJB consists of multiple "parts", inheritance is achievable in a rather limited fashion (see FAQ answer on inheritance here). There have been noteworthy suggestions on using multiple inheritance of the remote interface to achieve polymorphism, but the problem of how to share method signatures across **whole** EJBs remains to be addressed. The

following is one solution to achieving polymorphism with Session Beans. It has been tried and tested on WebLogic Apps Server 4.50 with no problems so far.
We will use an example to show how it's done. Say, there are 2 session beans, Tiger and Lion, that share some method signatures but provide different implementations of the methods.

- Â· AnimalHome and Animal are the home and remote interfaces. The signatures of the polymorphic methods are in Animal.
- Â· AnimalBean is the base implementation bean.
- Â· TigerBean and LionBean extend from AnimalBean. They may override the methods of AnimalBean, implementing different behaviors.
- Â· Deploy Tiger and Lion beans, specifying AnimalHome and Animal as their home and remote interfaces. Note that Tiger and Lion should have different JNDI lookup names.

85. What classes does a client application need to access EJB?

It is worthwhile to note that the client never directly interacts with the bean object but interacts with distributed object stubs or proxies that provide a network connection to the EJB container system.
The mechanhism is as follows.

1. The client uses the JNDI context to get a remote reference (stub) to the home object (the EJBHome).
2. It uses the home to get a remote reference (stub) to the EJBs remote object (the EJBObject)
3. It then invokes business methods on this remote object.

The client needs the remote interface, the home interface, the primary key(if it is an entity bean).
In addition to these, the client would need the JNDI factory implementation, and the remote and home stubs. In some EJB servers the Factory and/or stubs can be dynamically loaded at run time. In other EJB servers they must be in the classpath of the client application.

86. What is an EJB primary key? How is it implemented when the database doesn't have a primary key?

A primary key is an object that uniquely identifies the entity bean. According to the specification, the primary key must be unique for each entity bean within a container. Hence the bean's primary key usually maps to the PK in the database (provided its persisted to a database).
You may need to create a primary key in the database for the sake of referential integrity. This does not, however, mean you NEED a primary key in the database. As long as the bean's primary key (which maps to a column or set of columns) can uniquely identify the bean it should work.

87. What's an .ear file?

An .ear file is an "Enterprise Archive" file. The file has the same format as a regular .jar file (which is the same as ZIP, incidentally). The .ear file contains everything necessary to deploy an enterprise application on an application server. It contains both the .war (Web

Archive) file containing the web component of the application as well as the .jar file. In addition there are some deployment descriptor files in XML.

88. Can beans use stored procedures in a database?

Stored procedures can be used by session beans that access the database using JDBC and bean-managed entity beans that use JDBC to manage their own persistence. JDBC provides a call interface for using stored procedures. An example is provided below:

```
InitialContext cntx = new InitialContext( );
DataSource dataSource = (DataSource) cntx.lookup("java:comp/env/jdbc/mydatabase");
Connection con = dataSource.getConnection( );

CallableStatement storedProcedure = con.prepareCall("{? = call someprocedure [(?,?
```

89. Is method overloading allowed in EJB?

Yes you can overload methods.

90. How can JMS be used from EJB 1.1?

The same as any client would use JMS. At this point there is no integration, but it is planned for a future release of the EJB spec.

91. Can primary keys contain more than one field?

Yes, a primary key can have as many fields as the developer feels is necessary, just make sure that each field you specify as the primary key, you also specify a matching field in the bean class. A primary key is simply one or more attributes which uniquely identify a specific element in a database. Also, remember to account for all fields in the equals() and hashCode() methods.

92. How does Container Managed Persistence work with automatically generated database ID fields? Should I map the ID field explicitly or leave it unspecified?

In the Deployment Descriptor, map the normal fields appropriately, but don't specify the auto-id field as one of the container managed fields.

93. Let's assume I use a JavaBean as a go-between a JSP and an EJB, and have, say, 50 concurrent clients that need to access the EJB functionality. Will the JSP container actually instantiate 50 instances of the bean, or can it reuse a single instance to access the EJB?

- It depends on the scope you associate with the JavaBean. If you assign the bean with page (which is the default) scope or request scope, a new bean will be instantiated for each incoming request.
- If you assign the bean with session scope, you will still have 50 instances loaded in memory (assuming each incoming request is triggered by a distinct client), although some may have been instantiated from an earlier request from the same client. However, you may not want to use the session scope for a high-volume site as these beans will continue to reside in memory, long after the request has been serviced, consuming valuable resources until they are invalidated either explicitly or due to a session timeout.
- You can also assign the bean with application scope, in which case it is instantiated just once before being placed into the servlet context of the container. It can then be accessed at a later time, as long as the server is up and running. Although this may sound like an attractive proposition, do note that you will have to contend with significant multithreading issues. For instance, you'll have to ensure that the bean is accessed in a thread-safe manner from each of the JSP files. While you can do this using explicit synchronization from within the JSP file, do note that your application may take a significant performance hit because of this - especially if you expect tens or hundreds of concurrent clients accessing your pages.
- So, in short, your best bet may be to assign the bean with request scope.

94. What happens when two users access an Entity Bean concurrently?

Taken from Enterprise JavaBeans by Richard Monson-Haefel, "EJB, by default, prohibits concurrent access to bean instances. In other words, several clients can be connected to one EJB object, but only one client thread can access the bean instance at a time. If, for example, one of the clients invokes a method on the EJB object, no other client can access that bean instance until the method invocation is complete."
So, to answer your question, two users will never access an Entity Bean concurrently.

95. What's the reason for having two interfaces -- EJBHome for creating, finding & removing and EJBObject for implementing business methods. Why not have an single interface which supports both areas of functionality?

This design reflects the common "Factory" Design pattern. The EJBHome interface is the Factory that creates EJBObjects. EJBObject instances are the product of the factory. The reason for having two interfaces is because they are both responsible for different tasks.

The EJBHome is responsible for creating and finding EJBObjects, whilst the EJBObject is responsible for the functionality of the EJB.

96. Which fields in beans should be public?

All Container Managed Fields in an Entity Bean must be public.
Ejb 1.1 spec section 9.4.1 - "The fields must be defined in the entity bean class as public, and must not be defined as transient."

97. How do you implement callbacks in EJB?

If your client is an EJB, it can pass a reference to itself to the method of the bean that it is calling. The EJB can then call methods directly on that interface.
If your client is a Java client, your client requires some sort of object that will "listen" for call-backs. This could be either a CORBA or RMI object. Again, you could pass references to these objects to the EJB, which could then invoke methods on the references.

98. When should I use bean-managed transactions instead of specifying transaction information in the deployment descriptor?

The Sun J2EE EJB Guide says like this:

- Although beans with container-managed transactions require less coding, they have one limitation: When a method is executing, it can be associated with either a single transaction or no transaction at all. If this limitation will make coding your session bean difficult, you should consider using bean-managed transactions.
- The following pseudo-code illustrates the kind of fine-grained control you can obtain with bean-managed transactions. By checking various conditions, the pseudo-code decides whether to start and stop different transactions within the business method.

```
begin transaction
...
update table-a
...
if (condition-x)
commit transaction
else if (condition-y)
update table-b
commit transaction
else
rollback transaction
begin transaction
update table-c
commit transaction
...
```

I think what it means is there are some limitations in j2ee transaction support. In a container managed situation, nested or multiple transactions are not allowed within a method. if a biz method needs those features you need to go for bean managed transactions.

99. How do I automatically generate primary keys?

A common way to do it is to use a stateless session bean to retrieve the ID that you wish to use as the primary key. This stateless session bean can then execute an Oracle sequencer or procedure etc. to retrieve the ID value used as the primary key.

100. How is the passivation of Entity beans Managed?

The passivation of Entity beans is managed by the container. To passivate an instance, the container first invokes the ejbStore() method for synchronizing the database with the bean instance, then the container invokes the ejbPassivate() method. It will then return the bean instance back to the pooled state. (Every bean has an instance pool.)
There are two ways for transitioning an entity bean from the ready to the pooled state, by using the ejbPassivate() or ejbRemove() method. The container uses ejbPassivate() to disassociate the bean instance from the entity object identity, and uses ejbRemove() to remove the entity object.
When the instance is put back into the pool, it is no longer associated with an entity object identity. The container can now assign the instance to any entity object within the same entity bean home.
A bean instance in the pool can be removed by using unsetEntityContext().

101. To complete a transaction, which Transaction Attributes or Isolation Level should be used for a stateless session bean?

[For example, I have a method transfer which transfers funds from one account to another account.]
Vague question. Is the Session bean doing the DB work? I'll assume no.
Let's say AtmEJB is a Session bean with the transfer method. Let's say AccountEJB is an Entity bean.
Step 1:
When the client invokes the transfer method you want that to be the transaction; i.e. "the transfer transaction". therefore you need to set the tx attribute of transfer to something that will make the container start a tx at the beginning of transfer and terminate it at the end of transfer. RequiresNew might be a good choice but you need to look at all your use cases not just this one.
Step 2:
The AccountEJB methods invoked from the transfer method need to have a tx attribute that allows them to be part of an ongoing tx. That means that deposit and withdraw cannot be RequiresNew! (that would suspend the transfer tx and run in its own tx). Look at the spec for these: there are 3 that meets the criteria for deposit and withdraw in the transfer use case. Which one to use? What are the other use cases in which deposit and withdraw will be called? Find one that works for each one.

102. Explain the different Transaction Attributes and Isolation Levels with reference to a scenario.

The Enterprise JavaBeans model supports six different transaction rules:

- TX_BEAN_MANAGED. The TX_BEAN_MANAGED setting indicates that the enterprise bean manually manages its own transaction control. EJB supports manual transaction demarcation using the Java Transaction API. This is very tricky and should not be attempted without a really good reason.
- TX_NOT_SUPPORTED. The TX_NOT_SUPPORTED setting indicates that the enterprise bean cannot execute within the context of a transaction. If a client (i.e., whatever called the method-either a remote client or another enterprise bean) has a transaction when it calls the enterprise bean, the container suspends the transaction for the duration of the method call.
- TX_SUPPORTS. The TX_SUPPORTS setting indicates that the enterprise bean can run with or without a transaction context. If a client has a transaction when it calls the enterprise bean, the method will join the client's transaction context. If the client does not have a transaction, the method will run without a transaction.
- TX_REQUIRED. The TX_REQUIRED setting indicates that the enterprise bean must execute within the context of a transaction. If a client has a transaction when it calls the enterprise bean, the method will join the client's transaction context. If the client does not have a transaction, the container automatically starts a new transaction for the method. Attributes
- TX_REQUIRES_NEW. The TX_REQUIRES_NEW setting indicates that the enterprise bean must execute within the context of a new transaction. The container always starts a new transaction for the method. If the client has a transaction when it calls the enterprise bean, the container suspends the client's transaction for the duration of the method call.

TX_MANDATORY. The TX_MANDATORY setting indicates that the enterprise bean must always execute within the context of the client's transaction. If the client does not have a transaction when it calls the enterprise bean, the container throws the TransactionRequired exception and the request fails.

103. What is the most efficient approach for integrating EJB with JSP? Should the EJBs be invoked directly from within JSP scriptlets? Should the access take place from within Java beans? Or is it best to use custom tags for this purpose?

JSP scriptlet code should be minimal. Invoking EJB code directly on a JSP page results in many lines of code on your JSP page, including try...catch blocks to catch naming and finding exceptions.
Using a standard JavaBean as an intermediary between the JSP page and EJB server cuts down on the amount of code needed to add to a JSP page, and promotes reuse. The JavaBean should be a simple wrapper around the EJB you are accessing.
If you use a standard JavaBean you could also use the jsp:useBean tag to setup EJB parameters, such as the server URL and server security parameters.
Custom tags are also an option. However, they require a lot more coding than a simple JavaBean wrapper. The point should be to rewrite as little code as possible while at the same time keeping the JSP scriptlet content as light as possible.

104. How do you get a JDBC database registered with a JNDI name so that it can be accessed from an EJB?

The short answer is that it depends on which container you're using to some extent. The one thing that (should be) common in EJB 1.1 containers is that the ejb-jar.xml file's entry for that bean needs to contain a 'resource-ref' stanza, like so:

```
<resource-ref>
<res-ref-name>jdbc/LocalDB2<res-ref-name>
<res-type>javax.sql.DataSource<res-type>
<res-auth>Container<res-auth>
<resource-ref>
```

The res-ref-name is the most interesting part. This is the JNDI name relative to the java:comp/env namespace. Hence, to get this connection you'd do (in your bean):

```
Context context = new InitialContext();
DataSource source = context.lookup("java:comp/env/jdbc/LocalDB2");
```

which gives you a DataSource that you can call getConnection on.
The other half of this is container specific and done at deployment time by a 'Deployer' or 'Assembler' (to use the rolenames specified by the EJB spec.) This can work very differently from one container to the next, but here are a couple of (abbreviated) examples.
With Weblogic 5.1, you must define a connection pool in weblogic.properties, then edit the weblogic specific deployment descriptor (using the EJB Deployment tool) to associate the resource-ref specified in ejb-jar.xml with that connection pool.
With Inprise Application Server 4.0, all of the parameters for the connection (JDBC driver, connection URL, etc.) are specified in the inprise specific deployment descriptor (also editable via their deployment tool).
Other servers will have other ways of associating the resource-ref with a pre-defined connection pool.

105. How to manage fields that can have null values in a container-managed Entity bean?

First of all, let's just set up a typical scenario:
You are developing a product which allows a bank to sign up new customers online. You have done analysis and design and settled on having two tables: 'users' and 'account' (let's keep it simple). Each "user" will have a corresponding "account". The foreign key between the two will be the "account number".
So, for the 'users' table, you have the following fields: firstName, lastName, userId, password, and acctNum. When this table is created in the database, it is empty. Now you must relate your EJB code to this table for persistence. For simplicity sake I will leave out the Session bean (which I would use to talk to my Entity bean), the Entity bean primary key class, and the home and remote interfaces.
We have the UserBean:

```
public class UserBean implements javax.ejb.EntityBean {
public String firstName = null;
public String lastName = null;
public String userId = null;
```

```
public String password = null;
public long acctNum = -1;

/**
* Called by the container after the UserHome.create() is called
*/
public void ejbCreate(String userId, String password, long acctNum) {
this.userId = userId;
this.password = password;
this.acctNum = acctNum;
}
...
...

public void setUserData(UserData data) throws RemoteExeption, UserDataException {
this.firstName = data.getFirstName();
this.lastName = data.getLastName();
}
...
...
}
```

Now, assuming you have the User (remote interface class), UserHome (home interface class), UserPK (primary key class) already done, you need to create the bean's deployment descriptor. Inside the deployment descriptor, you must specify the database table, 'users', which this bean will map to. Also, you must specify which fields from your bean map to which fields in the 'users' database table. (This is how the container knows which fields to persist between your bean and the database table.) Now assuming all code compiles and you have an EJB server up and running and you have deployed your bean, all you need to do is write a client (I would use a client to access a session bean, say 'CustomerSession', which would talk to my entity bean) to create a new user and pass in the userId, password and acctNum values, which will create the new user in the database.
Notice the fields in the UserBean will now be set, but the firstName and lastName fields will still be set to null. These fields will still be empty in the database and will not change until these fields are set in the bean, since it is persisted. Now, call the setUserData(UserData data) method for setting the firstName and lastName, and these fields will now be set in the database and no longer be null. The container will handle the persistence of any fields which are set to null, just as it will handle any fields which are set to some meaningful value.

106. How can I debug my EJB applications?

This depends upon which EJB Container you are using.
Borland's JBuilder 3.5, Foundation Edition allows you to debug any Java 2 application, including the Inprise Application Server, WebLogic Server and J2EE Reference implementation. You can download it free from www.borland.com/jbuilder.
There are other IDE's out there including NetBeans/Forte www.sun.com/forte/ffj/ce/ that can also debug EJB.

107. How can an applet talk directly with EJB?

An applet must use the same procedure as any other Java class: it must use JNDI to locate the EJB Home Interface, then use RMI to talk with the Home Interface as well as the EJB itself.
This means that the J2EE and/or JNDI and/or RMI classes need to be present in the applet's Java Virtual Machine. The easiest way to assure this is to use the latest Java Plug-in. Netscape 6, aka Mozilla, ships with the Java Plug-in. Other browsers have various problems with RMI and JNDI classes that are beyond the scope of this answer.
Note, however, that it is not recommended to use EJB directly from applets, in part due to compatibility issues. Instead, you can use Servlets inside the application server to provide an HTML front-end that is assured to work on a much larger base of clients.

108. What is the best way of implementing a web application that uses JSP, servlet and EJB technologies all together following a Model View Controller (MVC) architecture?

[See the Sun J2EE Blueprints for "an integrated set of documentation and examples that describe and illustrate 'best practices' for developing and deploying component-based enterprise applications using the J2EE platform" including some good architecture whitepapers and source code. -Alex]
Hmm, 'Best Way' is a bit rough - there are several 'good' ways, and the usual set of trade-offs between them. (I'm a consultant - I have to start any answer with "It depends...", otherwise they revoke my whiteboard privileges)
The main thing you need to keep in mind as you design this sort of a system is that you want the interface into the EJB's to be rather narrow: in any flow, the ideal is to call one EJB method (hopefully on a stateless session bean), and let it make calls to entities on your behalf, then hand back the data you need to display.
How you display it depends: you can either embed beans on your JSPs and let the beans make that hopefully-one EJB call, or you can post to a servlet, let that make the call, then forward to the JSP for display. The second of these is more flexible and gives you more leverage to hide, change, and enforce your site's structure. The first, however, will be easier for developers new to this web thing to follow.
Essentially, I'm saying that Entity beans are your model, your controller is Session beans (maybe with a bit of help from servlets or beans), and your JSPs, beans and servlets are your View.
One thing to note here: this discussion strongly implies that your EJBs are capable of externalizing their state as some number of very simple 'value objects' (not EJBs themselves, just something we can pass back and forth). These value objects are probably tuned tightly to a workflow, and will be produced by that session bean. This way the traffic between the EJB (server) and the JSP/Servlet (client) is tuned to what the client needs, while the transaction load on the server is minimized.

109. When does the container call my bean's ejbCreate / ejbPostCreate / ejbStore / ejbPassivate / ejbActivate / ejbLoad / ejbPassivate method? And what should I do inside it?

The lifecycle of an enterprise bean is the heart of the EJB system. Your bean is basically implemented as a set of callback methods. There are a lot of these methods, which can be confusing; however, the implementation of each one is actually quite straightforward. Most of the time you can get away with implementing only a few of them.
Using Bean-Managed Persistence, each callback method ejbX means the obvious thing (usually "you must do X").
Using Container-Managed Persistence,

- ejbCreate means "fill in your defaults"
- ejbPostCreate means "your primary key is now valid; keep initializing"
- ejbStore means "I'm about to store your data into the DB"
- ejbPassivate means "I just stored your data, and I'm about to passivate you"
- ejbActivate means "I just activated you, and I'm about to load in your data"
- ejbLoad means "I just loaded your data from the DB"

110. What's the difference between EJBHome, EJB Home, EJB Object, EJBObject and EJB (not to mention Home Interface and Remote Interface)?

The EJB spec is all about really bad naming decisions.
First, an Enterprise JavaBean is not a JavaBean (but you already knew that).
The **"Home Interface"** is actually a Factory Object. It is responsible for locating or creating instances of the desired bean, and returning remote references.
When you write the source code for the EJB Home Interface, you must extend the interface **EJBHome**, and provide method signatures for all the desired create() and find() methods. An object that implements the Home Interface is automatically generated by the EJB Server tools.
The **"EJB Object"**, or Remote Object, is actually a Wrapper. It sits somewhere inside the container, between the client and your code. It is responsible for performing all the setup and shutdown tasks (like opening transactions, or restoring data state) immediately before and after your enterprise bean is called.
The "EJB Object" is generated by the EJB Server tools -- you don't have to write any part of it. However, you do have to write another interface, called the **"Remote Interface"** or the **"EJBObject Interface,"** that extends interface **EJBObject**, and provides method signatures for all the business methods. The server automatically generates a Java class that implements the Remote Interface; it is this object that is registered with RMI, and a reference to it is returned by the Home Interface (which we now know is actually a Factory Object).
The **"EJB,"** or Enterprise Bean, ironically, is **not** the EJB Object (even though it is an EJB and it is an object). It doesn't even **implement** the EJBObject interface, nor does it implement the Remote Interface. Instead, it implements either the EntityBean interface or the SessionBean interface. It also must implement all the methods defined in the Remote Interface -- but it doesn't actually implement the interface (in the Java sense). This is unfortunate, since we cannot rely on the Java compiler to make sure we've implemented all the right methods. It must also implement one ejbCreate() method for each create() method in the Home Interface (as well as ejbFind()/find()).

111. Is there a difference between container managed and bean managed persistence in terms of performance?

[Short answer: with bean-managed persistence, you can optimize your queries and improve performance over the generalized container-managed heuristics. But container-managed persistence is very convenient, and vendors will be working to improve its performance as time goes on. -Alex]
There is of course a difference as many CMPs use O-R mapping using metadata, which is slower than hardcoded queries (except vendors like GemStone that use a OODB which is slow anyway!) As always, a lot depends on the database schema. Given that CMP is still evolving, complex relationships (e.g.inheritance) and distributed transactions are not even supported by most EJB server vendors, leave alone performance.
Having said that however, it does not seem right to compare BMP and CMP on performance because the motivation of CMP is precisely to relieve bean providers from thinking about this! In (J2EE) theory, a good CMP implementation should perform well in a production environment; in practice, except for a couple of vendors who have traditionally been strong in persistent storage space (e.g. Persistence Software, GemStone) you will not find great CMP support at this very moment.
BMP offers a tactical approach while CMP is more strategic. Which implies that if you can work-around some (perhaps severe) limitations for near-term, there may be much to gain with CMP as the vendor offering matures.

112. Given that RMI-IIOP does not support distributed garbage collection (unlike RMI-JRMP), do I need to do something explicitly to GC beans, or is it magically handled by the EJB framework?

It is the Containers' responsibility to handle distributed garbage collection. This is how EJB's were designed.

113. OK, so EJB doesn't support user-created threads. So how do I perform tasks asynchronously?

If your EJB does not need to know about the results of the aynch calls, then you can use JMS to send an asynch. message to another part of the system.
Another alternative is to place the multithreaded code inside a CORBA or RMI server and call this from your EJB. Always keep site of the big picture, RMI and CORBA are part of J2EE and can be used as part of a 'J2EE' solution.
There are some things that these technologies can do that EJB at this present time cannot.

114. What is an EJB Context?

EJBContext is an interface that is implemented by the container, and it is also a part of the bean-container contract. Entity beans use a subclass of EJB Context called EntityContext. Session beans use a subclass called SessionContext. These EJBContext objects provide the bean class with information about its container, the client using the bean and the bean itself. They also provide other functions. See the API docs and the spec for more details.

115. Can I deploy a new EJB without restarting my server? (I'm using Weblogic.)

Sure. WebLogic Server4.5 includes "hot deploy" feature that allow you to deploy, redeploy or undeploy EJBs while the Server is running, from the Weblogic Console. Deployment of EJBs made through the console are however lost when you restart the WebLogic Server.

116. How to setup access control in an EJB such that different application clients have different rights to invoke different methods in one EJB?

1. Set up the different users/groups and the methods each can have access to in your deployment descriptor. Note: You don't have to specify different methods for each user, you could also just specify different users to your entire bean - for example if you only wanted another component of your application talking to your bean.
2. Inside your client code, whenever you make your connection to the EJB server (to look up the bean) you need to specify the user and password, in order to set the Identity of the client:

```
...
Properties p = new Properties();
..
p.put(Context.SECURITY_PRINCIPAL, "user");
p.put(Context.SECURITY_CREDENTIALS, "password");
...
```

3. Inside your bean, you can do "extra" security checks (if you used 'Role'-based security): (Assuming you have a 'manager' role defined in your deployment descriptor and a user assigned to this role)

```
public int getAccountBalance(accountId) {
if (ejbContext.isCallerInRole("manager"))
return balance;
}
```

   You could also enforce security to your EJB server. Using Weblogic, you could add the following to your weblogic.properties file:

```
...
weblogic.password.user=password
...
```

   where "user" is the username you grant access for and "password" (after '=') is the password for this username.

117. How is persistence implemented in enterprise beans?

Persistence in EJB is taken care of in two ways, depending on how you implement your beans: container managed persistence (CMP) or bean managed persistence (BMP).

For CMP, the EJB container which your beans run under takes care of the persistence of the fields you have declared to be persisted with the database - this declaration is in the deployment descriptor. So, anytime you modify a field in a CMP bean, as soon as the method you have executed is finished, the new data is persisted to the database by the container.
For BMP, the EJB bean developer is responsible for defining the persistence routines in the proper places in the bean, for instance, the ejbCreate(), ejbStore(), ejbRemove() methods would be developed by the bean developer to make calls to the database. The container is responsible, in BMP, to call the appropriate method on the bean. So, if the bean is being looked up, when the create() method is called on the Home interface, then the container is responsible for calling the ejbCreate() method in the bean, which should have functionality inside for going to the database and looking up the data.

118. Can the primary key in the entity bean be a Java primitive type such as int?

The primary key can't be a primitive type--use the primitive wrapper classes, instead. For example, you can use java.lang.Integer as the primary key class, but not int (it has to be a class, not a primitive).

119. How do I map a Date/Time field to an Oracle database with CMP?

[Question continues: (I have written a wrapper class with the help of java.util.GregorianCalendar but it doesn't store time in Oracle database, whereas it stores Date without any problem.)]
Use the java.sql.Timestamp field to store your Date/Time in your entity bean, and then declare the corresponding field as 'Date' type in the Oracle database and it will work fine. You will have the "date" value and the "time" value preserved.

120. What is the difference between a Server, a Container, and a Connector?

To keep things (very) simple:
An EJB server is an application, usually a product such as BEA WebLogic, that provides (or should provide) for concurrent client connections and manages system resources such as threads, processes, memory, database connections, network connections, etc.
An EJB container runs inside (or within) an EJB server, and provides deployed EJB beans with transaction and security management, etc. The EJB container insulates an EJB bean from the specifics of an underlying EJB server by providing a simple, standard API between the EJB bean and its container.
(Note: The EJB 1.1 specification makes it clear that it does not architect the interface between the EJB container and EJB server, which it says it left up to the vendor on how to split the implementation of the required functionality between the two. Thus there is no clear distinction between server and container.)
A Connector provides the ability for any Enterprise Information System (EIS) to plug into any EJB server which supports the Connector architecture.

121. What is "clustering" in EJB?

Clustering refers to the ability of multiple load-balanced web servers to share session and entity data. It is a major feature of web application servers. Standardized support for clustering was one of the primary motivations behind the EJB spec.
Clustering also applies to Servlet containers sharing HttpSession data (similar to EJB Session Beans).

122. What is "hot deployment" in WebLogic?

"Hot Deployment" in weblogic is the act of deploying, re-depolying, and un-deploying EJBs while the server is still running (you don't have to shutdown the server to deploy an EJB).

123. Can I specify specific WHERE clauses for a find method in a CMP Entity Bean?

The EJB query language is totally vendor specific in EJB1.1. It is being standardized in 1.2. Yes, you can specify the where clause for a find method. This is the example for EJB's deployed on weblogic:

```
findBigAccounts(double balanceGreaterThan): "(> balance $balanceGreaterThan)"
```

where balance maps to some field in the table.

124. When using a stateful session bean with an idle timeout set, how can the bean receive notification from the container that it is being removed due to timeout?

[Question continues: ? (Through some tests, it looks like none of the standard EJB callback methods are called when a stateful session bean is removed due to idle-timeout.)]
According to the spec, ejbRemove need not (or must not) be called in this case. ejbPassivate is simply the Wrong Thing to be called (the bean is transitioning to the 'does not exist' state, not the 'passive' state).
The EJB 1.1. spec says in section 6.6.3 Missed ejbRemove Calls:

- The application using the session bean should provide some clean up mechanism to periodically clean up the unreleased resources.
- For example, if a shopping cart component is implemented as a session bean, and the session bean stores the shopping cart content in a database, the application should provide a program that runs periodically and removes “abandoned” shopping carts from the database.

Probably not the answer you're looking for, especially if you allocate some other resource (a Message Queue, for example) that you need to release. Although, if you're using a resource, you really should be getting it when you need it (via JNDI) and returning it back to the pool right away.

125. I have created a remote reference to an EJB in FirstServlet. Can I put the reference in a servlet session and use that in SecondServlet?

Yes.
The EJB client (in this case your servlet) acquires a remote reference to an EJB from the Home Interface; that reference is serializable and can be passed from servlet to servlet. If it is a session bean, then the EJB server will consider your web client's servlet session to correspond to a single EJB session, which is usually (but not always) what you want.

126. What is the difference between a Component Transaction Monitor (CTM) and an Application Server?

A *Component Transaction Monitor* (CTM) is an application server that uses a server-side component model. Since a CTM is a *Transaction Processing* monitor (TP), it is expected to provide services for managing transactions, security, and concurrency. In addition, CTMs also facilitate distributed object architectures and provide facilities for object persistence. In short, a CTM is a specific type of application server.

127. How can I call one EJB from inside of another EJB?

Just do it!
EJBs can be clients of other EJBs. It just works. Really. Use JNDI to locate the Home Interface of the other bean, then acquire an instance reference, and so forth.

128. When using Primary Keys, why do I have to implement the hashCode() and equals() method in my bean?

Implementing the hashCode() and equals() functions ensure that the primary key object works properly when used with hash tables. Hash tables are the preferred way EJB servers use to store and quickly retrieve instantiated entity beans.
If session #1 uses widget "A" (which is an entity bean) then the server needs to do some work to instantiate and initialize the object. If session #2 then requests the same widget "A", the EJB server will look in its hash table of existing entity beans of type widget to see if widget "A" has already been instantiated.

129. Can I deploy two beans in a single jar file? If so, how?

Yes, multiple EJBs can be deployed in a single jar file. The deployment is somewhat different between EJB 1.0 and EJB 1.1.
In EJB 1.1 and in the draft EJB 2.0 specification, instead of a manifest and serialized deployment descriptors there is a single shared XML deployment descriptor named META-INF/ejb-jar.xml. Within ejb-jar.xml there must be either a <session> or <entity> element for each bean in the jar file. For example, the following XML fragment is for a jar file that contains one entity and one session bean:

```
<ejb-jar>
<enterprise-beans>
<session>
<ejb-name>MySessionBean</ejb-name>
... other xml elements describing the bean's deployment properties ...
</session>
<entity>
<ejb-name>MyEntityBean</ejb-name>
... other xml elements describing the bean's deployment properties ...
</entity>
</enterprise-beans>
</ejb-jar>
```

The EJB 2.0 draft specification for deployment descriptors differs from EJB 1.1 only in the addition of XML elements for describing additional bean properties.

130. Why use EJB when we can do the same thing with servlets?

Actually, servlets/JSPs and EJB are complementary, not competing technologies: Servlets provide support for writing web based applications whereas EJBs provide support for writing transactional objects. In larger web systems that require scalability, servlet and JSP or XML/XSL technologies provide support for the front end (UI, client) code, where EJB provides support for the back end (database connection pooling, declaritive transactions, declaritive security, standardized parameterization...)
The most significant difference between a web application using only servlets and one using servlets with EJBs is that the EJB model mandates a separation between display and business logic. This is generally considered a Good Thing in non-trivial applications because it allows for internal reuse, allows flexibility by providing a separation of concerns, gives a logical separation for work, and allows the business logic to be tested separately from the UI (among others).
Some of the hings that servlets and JSPs can do that EJBs cannot are:

- Respond to http/https protocol requests.
- (With JSP) provide an easy way to format HTML output.
- Easily associate a web user with session information

Some of the things that EJBs enable you to do that servlets/JSPs do not are:

- Declaritively manage transactions. In EJB, you merely specify whether a bean's methods require, disallow, or can be used in the context of a transaction. The EJB container will manage your transaction boundaries appropriately. In a purely servlet architecture, you'll have to write code to manage the transaction, which is difficult if a logical transaction must access multiple datasources.
- Declaritively manage security. The EJB model allows you to indicate a security role that the user must be assigned to in order to invoke a method on a bean. In Servlets/JSPs you must write code to do this. Note, however that the security model in EJB is sufficient for only 90% to 95% of application code - there are always security scenarios that require reference to values of an entity, etc.

131. What restrictions are imposed on an EJB? That is, what can't an EJB do?

From the spec:

- An enterprise Bean must not use read/write static fields. Using read-only static fields is allowed. Therefore, it is recommended that all static fields in the enterprise bean class be declared as final.
- An enterprise Bean must not use thread synchronization primitives to synchronize execution of multiple instances.
- An enterprise Bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard.
- An enterprise bean must not use the java.io package to attempt to access files and directories in the file system.
- An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast.
- The enterprise bean must not attempt to query a class to obtain information about the declared members that are not otherwise accessible to the enterprise bean because of the security rules of the Java language. The enterprise bean must not attempt to use the Reflection API to access information that the security rules of the Java programming language make unavailable.
- The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams.
- The enterprise bean must not attempt to set the socket factory used by ServerSocket, Socket, or the stream handler factory used by URL.
- The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread; or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups.
- The enterprise bean must not attempt to directly read or write a file descriptor.
- The enterprise bean must not attempt to obtain the security policy information for a particular code source.
- The enterprise bean must not attempt to load a native library.
- The enterprise bean must not attempt to gain access to packages and classes that the usual rules of the Java programming language make unavailable to the enterprise bean.
- The enterprise bean must not attempt to define a class in a package.
- The enterprise bean must not attempt to access or modify the security configuration objects (Policy, Security, Provider, Signer, and Identity).
- The enterprise bean must not attempt to use the subclass and object substitution features of the Java Serialization Protocol.

The enterprise bean must not attempt to pass this as an argument or method result. The enterprise bean must pass the result of SessionContext.getEJBObject() or EntityContext. getEJBObject() instead.

132. Why do we have a remove method in both EJBHome and EJBObject?

With the EJBHome version of the remove, you are able to delete an entity bean without first instantiating it (you can provide a PrimaryKey object as a parameter to the remove method). The home version only works for entity beans. On the other hand, the Remote interface version works on an entity bean that you have already instantiated. In addition, the remote version also works on session beans (stateless and statefull) to inform the container of your loss of interest in this bean.

133. Why is it that business methods should not be declared final?

I believe that the basic reason is that mandating non-final business methods allows container developers to implement their EJB container via inheritence. They can generate a class that extends your bean, with methods that perform transactional housekeeping, then call the inherited method (which is the one you wrote in your bean), then perform more housekeeping.
That said, I know of no major container that does things this way (although some of the OODBMS vendors may)

134. Why is ejbFindByPrimaryKey mandatory?

An Entity Bean represents persistent data that is stored outside of the EJB Container/ Server.
The **ejbFindByPrimaryKey** is a method used to locate and load an Entity Bean into the container, similar to a **SELECT** statement in SQL.
By making this method mandatory, the client programmer can be assured that if they have the primary key of the Entity Bean, then they can retrieve the bean without having to create a new bean each time - which would mean creating duplications of persistent data and break the integrity of EJB.

135. How can I pass init parameters to enterprise beans?

You can specify Environment Entries that are accesssible by your EJB's. Inside your **ejb-jar.xml** you define the environment entries.

```
<env-entry>
<env-entry-name>theParameter</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>theValue</env-entry-value>
</env-entry>
```

You can access the variable inside your EJB using the Environment Naming Context (in EJB 1.1)

```
Context ctx = new InitialContext();
String val = (String)ctx.lookup("java:comp/env/theParameter");
```

136. Should I use CMP or BMP for an application with complex data manipulation & relations?

Generally, you should use CMP unless you're forced to use BMP due to limitations of the mapping tools. Also, "complex" is relative; some relatively complex data models can be captured with mapping tools, but some cannot.

137. For session beans, we can use the SessionSynchronization interface. For entity beans, how do we have control over a transaction?

The SessionSynchronization interface is used by the Session beans to Synchronize the Instance variables after a rollback or after a commit operation, because container does not have any other way to inform the bean of these operations.
With Entity beans, this is not a problem as the Container automatically calls the ejbLoad method that refreshed the values from the database.