Table of Contents

11 security in EJB

Protect Your Secrets

... no way! You can't let Susie hear this. I better put some method permissions in my DD to stop her from calling! Let's put her in the 'crybaby' security role...

Keep your secrets. Security is about **authentication** and **authorization**. First, you have to prove your identity, and then we'll tell you what you're allowed to do. Security is easy in EJB, because you're only dealing with *authorization*. You decide *who* gets to call which *methods* on your beans. Except one problem... if you're a Bean Provider or App Assembler, you probably don't *know* who the users are going to be! So you make stuff up. You make up roles, like job titles, including Manager, Supervisor, Admin, etc. and when someone deploys your application in a real company, that Deployer maps between your made-up names (Manager) and *real* people who will use the app.

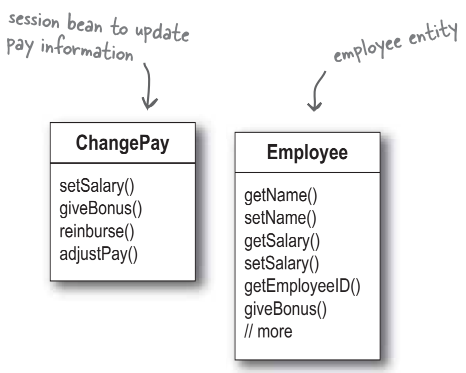
OBJECTIVES

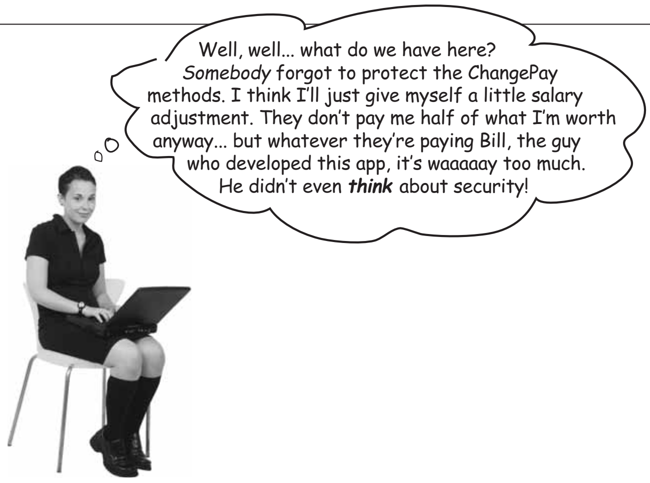
Security

	Official:	*What it really means:*
14.1	Identify correct and incorrect statements about the EJB support for security management including security roles, security role references, and method permissions.	You have to know that you can do two kinds of security in EJB: programmatic and declarative. Declarative means your security is defined in the Deployment Descriptor. You need to know that security in EJB is mostly about declaring a set of abstract security roles, then declaring which methods each of those roles is allowed to call.
14.2	From a list of responsibilities, identify which belong to the Application Assembler, Bean Provider, Deployer, Container Provider, or System Administrator.	Pay close attention to this in the chapter. You need to think about the whole process in a component-based development way, to know who is responsible for what. You need to know that the Bean Provider usually won't have any security responsibilities, but that if he does use the isCallerInRole() method, then he must tell the App Assembler what name he hard-coded in as the argument to the isCallerInRole() method, by putting a security role reference in the DD. You must know that the App Assembler's job is to define security roles, method permissions, and role-links mapping from the programmer's made-up role references and the App Assembler's real security roles. Finally, you must know that the Deployer is responsible for mapping (in a vendor-specific way) between real users and groups and the App Assembler's abstract security roles, and that this is done using Container tools or property files.
14.3 & 14.4	Given a code listing, determine whether it is a legal and/or appropriate way to programmatically access a caller's security context. Given a security-related deployment descriptor tag, identify correct and incorrect statements and/or code related to that tag.	You have to know everything about the DD tags related to security, including that security role references, and role links, are in the <enterprise-bean> section, but security roles and method permissions are in the <application-assembly> part. You must know how the <security-identity> tag is used, and that message-driven beans must never <use-caller-identity>! You need to know that the two programmatic security methods are called on a session or entity bean's context.

Imagine you're writing a payroll application...

This will be, without a doubt, the best payroll program ever written. I'm gonna RULE the IT department when they see this app. Even *I* am still astonished by my superior coding skills...




Well, well... what do we have here? *Somebody* forgot to protect the ChangePay methods. I think I'll just give myself a little salary adjustment. They don't pay me half of what I'm worth anyway... but whatever they're paying Bill, the guy who developed this app, it's waaaaay too much. He didn't even ***think*** about security!

What can you do?

Who really is responsible for making the program secure? How does it happen? There's good news and bad news with EJB security.

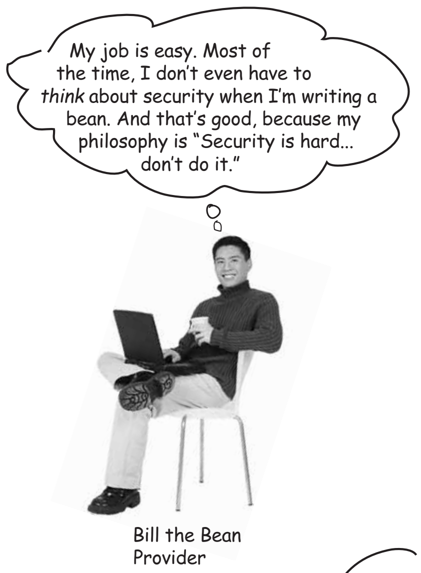
1 Security in EJB is easy. *It's about AUTHORIZATION.*

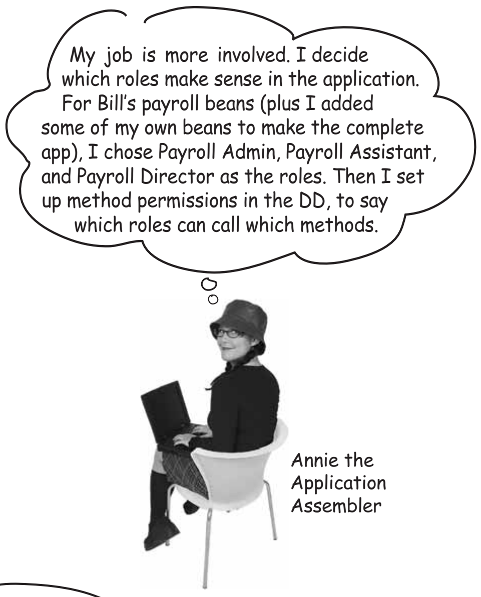
- Most of the time, ***you don't put any security-related code into your bean classes.***
- In fact, the ***Bean Provider does not even think about security***, except in very special cases we'll look at in the last part of this chapter.
- Security in EJB is about saying WHO can call WHAT. ***You can restrict access for each individual method in your application, to only those individuals that are in a privileged 'role'*** (Director, Payroll Admin, Payroll Assistant, etc.)
- The EJB part of the security—the part that specifies the roles, and the methods those roles can access—is done ***declaratively***, ***in the deployment descriptor,*** using simple XML tags.
- At the EJB level, ***it really is as simple as saying, "The setSalary() method can be accessed by ONLY Directors and Payroll Administrators."***

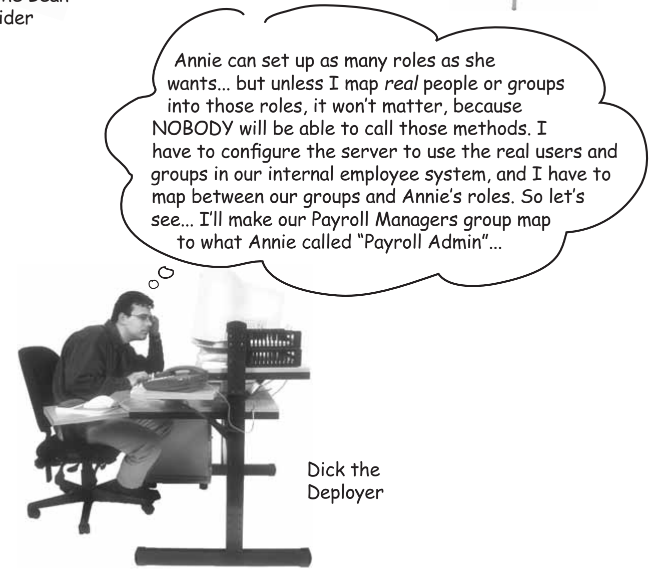
2 Security in EJB is abstract. *It's NOT about AUTHENTICATION.*

- The EJB spec makes it very easy to define roles, and to assign roles to methods in order to control access to the methods. But ***the spec says NOTHING about how the system will KNOW which real human beings belong to those roles.*** Somewhere outside of the EJB deployment descriptor (and outside the specification) you still have to say that Jack O'Bryan is in the Director role (and probably other roles as well). Or say that all Payroll Managers in company XYZ are qualified to be in the role of what the application has labeled Payroll Admin.
- The spec also ***does not define how a real human being can be authenticated*** to the system. In other words, there's nothing in EJB about how you will know that, say, the person logging in as Jack O'Bryan really *is* Jack O'Bryan. This might happen in a variety of ways, although one typical option is to have clients authenticate through the web tier (they log in using a browser) of a J2EE-friendly web server.
- The security in the real operating environment—the company running the application—must have a security structure in place, and in a format that your specific EJB server can hook into. Or, at the very least, you need a server that lets you specifically configure a security realm. Even if it's nothing more than a list of names, passwords, and the 'roles' those names belong to, you still need *some* infrastructure outside of EJB.

How to do security in EJB


My job is easy. Most of the time, I don't even have to *think* about security when I'm writing a bean. And that's good, because my philosophy is "Security is hard... don't do it."
Bill the Bean Provider


My job is more involved. I decide which roles make sense in the application. For Bill's payroll beans (plus I added some of my own beans to make the complete app), I chose Payroll Admin, Payroll Assistant, and Payroll Director as the roles. Then I set up method permissions in the DD, to say which roles can call which methods.
Annie the Application Assembler


Annie can set up as many roles as she wants... but unless I map *real* people or groups into those roles, it won't matter, because NOBODY will be able to call those methods. I have to configure the server to use the real users and groups in our internal employee system, and I have to map between our groups and Annie's roles. So let's see... I'll make our Payroll Managers group map to what Annie called "Payroll Admin"...
Dick the Deployer

The Application Assembler's job: access control

The App Assembler knows the application. She knows what the methods do, and how they're supposed to be used. The App Assembler knows the abstract roles that make sense logically, for this application. For example, she knows that there's no need for a Marketing role in the payroll application. But she knows that there should be a least three levels of access for the app:

1. People who have full control and can both view and change an employee's payroll data.

2. People who can read everything and modify some things.

3. People who can read some things, but can't modify anything.

So her job is broken into two parts:

1 Define the roles

- Figure out which roles makes sense in the application, and come up with names for these roles. Since the App Assembler might not be working in the real environment where the application will run, she's just *making up abstract names*. In other words, her names don't have to correspond to anything in the real world. For all we care, she could name the three roles Clown, Mime, and Juggler. As long as she can describe them well enough for the deployer to figure out which real people belong to those roles, it doesn't matter that the names are made up.

- In the deployment descriptor, define a **`<security-role>`** element for each role in the application.

- In the deployment descriptor, use the **`<role-name>`** element to define the made-up name for this role.

2 Assign method permissions

- Figure out the methods from the client view to which each of the roles should have access. Remember, the client view includes not only the methods the Bean Provider declared in the home and component interface of the bean, but also the methods of the super interfaces EJBObject, EJBHome, or EJBLocalObject, and EJBLocalHome.

- In the deployment descriptor, define a **`<method-permission>`** element that lists one or more security roles and one or more methods.

- If she chooses, the App Assembler can can use the **`<unchecked/>`** element to indicate that a particular method doesn't require any authorization (in other words, *anybody* can call it).

- If she chooses, she can use the **`<exclude-list>`** element to define a list of methods that nobody can call, ever. In other words, no role will be able to call the methods on the exclude list.

there are no Dumb Questions

Q: You mention an exclude-list. If nobody will be allowed to call the method, then why is the method there?

A: Remember, beans are reusable components, meant to be assembled in multiple applications. A method that makes sense in one application might not make sense in another. For example, a ProductInventory bean that represents products in a database might have methods for an inventory administrator to add products or change prices, etc., but a client catalog program using that same bean should never need to see anything but the *read-only* methods (i.e. just the *getters*).

Q: I would think that the person writing the beans would be the one who REALLY knows what the roles should be.

A: Remember, beans are reusable components. Yes, we just said that in the previous question, but it works here too. The Bean Provider should not know *exactly* how these beans are being used. If he gets too specific when he designs the code and writes the beans, then the beans probably won't work well outside the specific payroll application that he has in mind.

Only the Application Assembler knows for sure how the different pieces of the application related to one another, and which roles make sense. There is one small exception to this that we'll talk about later in the chapter, when we look at 'programmatic security'.

Sharpen your pencil

Look at the following applications, and try to come up with security roles that might make sense for these applications. Later, we'll figure out which methods each of the roles should be allowed to call.

1. An inventory system for an online store, keeps track of products sold and shipped, inventory levels, and prices.

2. An application for reserving and booking dude ranch vacations, through a travel agent.

3. A rule-based product recommendation system, that asks the user a series of questions, and then gives appropriate advice on which product the user should purchase.

4. An online match-making / personal ad service.

Defining the roles

The <security-role> element

Create the security roles in the <assembly-descriptor> part of the deployment descriptor. Remember, the deployment descriptor has two parts:

1. The <enterprise-beans> section that's created by the Bean Provider, and describes each bean in the application.

2. The <assembly-descriptor> part that's created by the App Assembler, and describes characteristics of the application as a whole, including the ways in which beans refer to one another, most security information, and transaction attributes.

```
<assembly-descriptor>

   <security-role>
       <role-name>Payroll Director</role-name>
   </security-role>

   <security-role>
       <role-name>Payroll Admin</role-name>
   </security-role>

   <security-role>
       <role-name>Payroll Assistant</role-name>
   </security-role>
...
</assembly-descriptor>
```

the only element you must have in a <security-role> element is <role-name>

Defining the roles... a better way

The <security-role> element with descriptions

Descriptions are optional elements for the <security-role> element.
But think about it.

Think about the poor, overworked, underpaid deployer (which in reality, is probably you since chances are good that you're wearing all three hats: Bean Provider, App Assembler, and Deployer—and what the heck, probably some Sys Admin in there too).

He's not a mind-reader.

What's the difference between Payroll Admin and Director? What was she thinking???

descriptions make it MUCH easier for the deployer to map the made-up abstract roles to the REAL groups in the company where the application will run.

```
<assembly-descriptor>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view and change all employee payroll
          information.
      </description>
      <role-name>Payroll Director</role-name>
   </security-role>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view all employee information, and
          change some pieces
      </description>
      <role-name>Payroll Admin</role-name>
   </security-role>

   <security-role>
      <description>
          This role is for the employees who have the
          power to view all employee payroll
          information.
      </description>
      <role-name>Payroll Assistant</role-name>
   </security-role>
...
</assembly-descriptor>
```

Defining the method permissions

The <method-permission> element

Everything in a bean's home and component interface is potentially callable by clients. Now that the App Assembler has defined the roles, she can define which methods each role is allowed to call. As she did with the security role definitions, she'll put the method permissions in the **`<assembly-descriptor>`** section of the deployment descriptor.

```
<assembly-descriptor>

   <method-permission>
       <role-name>Payroll Director</role-name>

       <method>
          <ejb-name>ChangePay</ejb-name>
          <method-name>*</method-name>
       </method>

       <method>
          <ejb-name>Employee</ejb-name>
          <method-name>*</method-name>
       </method>

   </method-permission>


   <method-permission>
       <role-name>Payroll Admin</role-name>

       <method>
          <ejb-name>ChangePay</ejb-name>
          <method-name>reimburse</method-name>
       </method>

       <method>
          <ejb-name>ChangePay</ejb-name>
          <method-name>giveBonus</method-name>
       </method>

       <method>
          <ejb-name>Employee</ejb-name>
          <method-name>getSalary</method-name>
       </method>
     . . .
   </method-permission>
  . . .
</assembly-descriptor>
```

A method permission starts with a role, and then says which methods in which beans this role is allowed to call

this method permission says, "A client in the role of 'Payroll Director' is allowed to call ALL methods in the ChangePay bean's client view (i.e. the home and component interfaces) and ALL methods in the Employee bean's client view.

this permission says "A client in the role of 'Payroll Admin' can call the reimburse and giveBonus methods of the ChangePay bean, and the getSalary method of the Employee bean.

Defining the method permissions

Three different ways to specify methods

You just saw two ways to specify a bean's method name: the wildcard (*) which means ALL methods in the bean, and the actual name of the method. But the name alone isn't always enough. We talked about this before—in the transactions chapter we faced the same problem when we had to specify transaction attributes. What happens if the method is overloaded?

Chances are, your design will treat all versions of an overloaded method in the same way. But there's an optional `<method-params>` element just in case you want, say, a particular security role to have permission for only *one* version of an overloaded method, but not the others.

① By wildcard (*) for ALL methods

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>*</method-name>
</method>
```

an asterisk (*) is the wildcard that means ALL methods in the bean's interfaces

② By name alone, for all methods with this name, regardless of arguments or whether they're in the home or component interface

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>takeOver</method-name>
</method>
```

this means that ALL overloaded methods named 'takeOver' will be accessible to the role.

③ By name and arguments, to distinguish between overloaded methods

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-name>takeOver</method-name>
    <method-params>
        <method-param>String</method-param>
    </method-params>
</method>
```

this specifies ONLY the takeOver method that takes a String, but no other overloaded versions.

Method permissions interact with one another as a union!

Method permissions do not relate to one another in the same way that transaction attributes do. With transaction attributes, using a wildcard says, "Every method in this bean will have this attribute unless I say otherwise, by naming a specific method in a different container-transaction tag." In other words, naming a specific method overrides the wildcard setting.

But with method permissions, using the wildcard says, "All methods in this bean can be accessed by this role." Nothing else you do in any other method permission will change that.

Transaction attributes

```
<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name> * </method-name>
   </method>
   <trans-attribute>RequiresNew</trans-attribute>
</container-transaction>

<container-transaction>
   <method>
      <ejb-name>BigBean</ejb-name>
      <method-name>useOldDatabase</method-name>
   </method>
   <trans-attribute>NotSupported</trans-attribute>
</container-transaction>
```

This says that all methods in BigBean will use the RequiresNew attribute, EXCEPT the useOldDatabase method, which will use the NotSupported attribute.

The second <container-transaction> overrides the wildcard one, and takes the useOldDatabase method *out* of the *RequiresNew* list and moves it to the *NotSupported* list.

Method permissions

```
<method-permission>
   <role-name>Minion</role-name>

      <method>
        <ejb-name>WorldDomination</ejb-name>
        <method-name>*</method-name>
      </method>
</method-permission>


<method-permission>
   <role-name>Boss</role-name>

      <method>
        <ejb-name>WorldDomination</ejb-name>
        <method-name>learnPlan</method-name>
      </method>
</method-permission>
```

This says that the role Minion can access ALL methods of WorldDomination.

AND that the Boss role can access ONLY the learnPlan method of WorldDomination.

The second <method-permission> adds to the first.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Watch out for <unchecked/>

Think of <unchecked/> as standing in for a <role-name> in a method permission. But because of the way method permissions interact, if you have a method permission defined for a method using <unchecked/>, it won't matter what other method permissions you set up for that method. One little <unchecked/> and it means the method is free for anyone to call, regardless of their principal or security role!

Method permissions with <unchecked/>

```
<method-permission>

   <role-name>AccountManager</role-name>

      <method>
        <ejb-name>Account</ejb-name>
        <method-name>increaseLimit</method-name>
      </method>
</method-permission
```

This says that AccountManager is allowed to call the increaseLimit() method on the Account bean.

```
<method-permission>

   <unchecked/>

      <method>
        <ejb-name>Account</ejb-name>
        <method-name>*</method-name>
      </method>
</method-permission
```

<unchecked/> goes in place of the <role-name> element

But who cares??!! By saying that there is a method-permission for Account that says ALL methods are unchecked, it doesn't matter what else we say about the Account bean's methods. They are all unchecked!

The <unchecked/> element overrides ALL other method permissions for a method.

Wait... I just thought of something else. Suppose I have the same method name in BOTH the home and component interface? And I want them to have different access controls?

How to deal with a bean that has two methods of the same name, but one is in the home interface and the other is in the component interface.

(Yes, we lied, there are actually *four* ways to describe a method: by wildcard (*), by name, by name and parameters, and by name and interface)

```
<method>
    <ejb-name>WorldDomination</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>takeOver</method-name>
</method>
```

The value of <method-intf> must be one of these four:

```
<method-intf>Remote</method-intf>

<method-intf>Home</method-intf>

<method-intf>Local</method-intf>

<method-intf>LocalHome</method-intf>
```

The Deployer's job: mapping actual humans to abstract roles

The App Assembler knows the *application*, but the Deployer knows the *operational environment*. We (and the spec) use the term *operational environment* as a fancy way of saying, *the business where the application is running*. Maybe the company bought the app off-the-shelf. Maybe they built it in-house. Doesn't matter. The Deployer *works* there. He knows the place. Most importantly, he knows how security is managed at the company (for example, the company might have all the employee names and passwords as part of an LDAP system). He's the best person to know how the abstract roles the App Assembler put in should map to real people and groups in his company.

He has two main jobs:

1 Assigning the security domain and principal realm to the app

- The company where the app is running has real employees. Somehow, those employees have a way of authenticating themselves to a server, probably with a name and a password. The security information in the operational environment has to be configured into the server, in such a way that the server can tell *who* is actually calling the method.

This happens outside of the EJB specification! In other words, it's vendor-specific.

2 Mapping users and/or groups to the abstract security roles

- The App Assembler made up the abstract security roles that best fit the payroll app. But those roles don't mean anything in the company where the app is going to run. The Deployer has to map between what Annie (the App Assembler) defined: Payroll Director, Payroll Admin, Payroll Assistant, to what *really* exists in the Deployer's company: Payroll Manager, Payroll Supervisor, HR Admin.

This, too, happens outside of the EJB specification, in a vendor-specific way.

Note: the spec uses the term "logical security roles" to refer to what we're calling the "abstract security roles", but we like the term "abstract". Just know that "logical security roles" and "abstract security roles" mean the same thing.

Relax

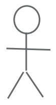
For the exam, you don't need to know HOW the Deployer does this, only that he MUST do these things.

Remember, the exam doesn't expect you to know about any vendor-specific functionality. But you DO have to know that it is the Deployer's responsibility to do this level of mapping between real humans and the App Assembler's abstract roles.

Principals and Roles, Users and Groups

Dan Johnsson authenticates to the system (using his login name and password) and is now represented by an instance of type Principal. In a vendor-specific way, the Deployer has told the EJB application that Dan's Principal is associated with one or more roles. So whenever Dan, as the client, calls a method on one of the bean's client interfaces, his Principal (and the roles to which that Principal has been assigned) propagates with the call. That means every call on the (conceptual) call stack will see Dan's Principal as the caller.

In EJB



abstract actor in the system (usually maps to a person)

① Principal

The client, authenticated to the system, is represented by a java.security.Principal object. This Principal is an abstract representation of some *thing* associated with a name, but there's no guarantee that the Principal name matches the login name of the client. It all depends on how your system handles security authentication. And don't be too attached to thinking that a Principal is always a unique individual. Sometimes a Principal represents a larger group like, say, SysAdmin.The Principal is associated with one or more abstract security roles that the App Assembler defined. About the only useful thing a bean can do with a Principal object is get its name (aPrincipal.getName()), but that's risky, because you won't be able to know exactly what that name represents unless you know the exact environment in which the bean is running. Yuck.

Manager
KeyAccount
Customer
PayrollDirector
SwedishTeam
SwingDancers

② Role

In EJB, a security role is an abstract role defined by the App Assembler, and although it doesn't match anything in the real world, the Deployer will map real users and groups in the company where the application is running, to the abstract roles. For example, the "Employee" abstract role might map to the "Slaves" group in the Deployer's company.

In the operational environment

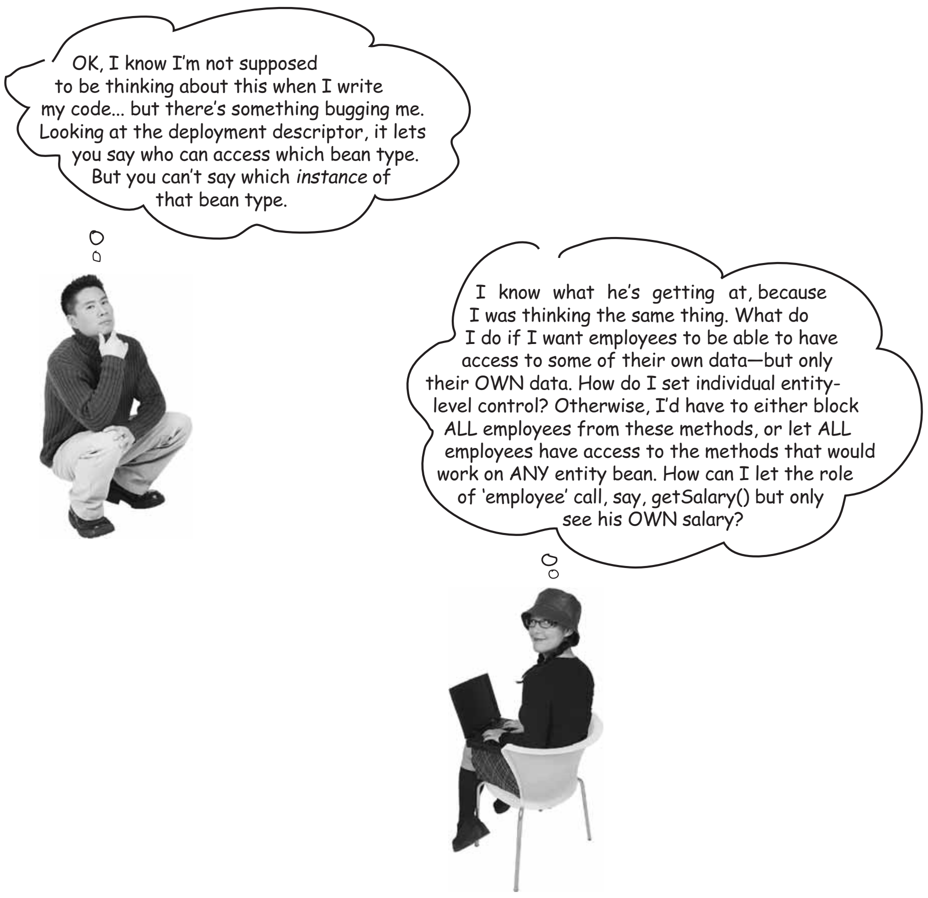
③ Users

Users are people in the real environment. Real users. Living, breathing, humans. They're represented in the real environment in some way, but of course in EJB we have no way of knowing in advance that, say, the company uses an LDAP server to hold the user security information. Typically, users are mapped to a log-in name and password, and that information is stored somewhere, and if you're lucky and make the right server decision, your EJB server can be configured directly to the system holding your security info. Pretty much the last thing you want to do is sit there and type 10,000 user names and passwords into your EJB server's property files, simply because your EJB server wasn't compatible with whatever your company is using to store user security information.

Customer
PayrollManager
Sweden
SalsaDancers

④ Groups

Typically, security information organizes users into one or more groups, often by department or location. The Deployer can often map directly from groups to abstract security roles, but he might also have to map individual users.


OK, I know I'm not supposed to be thinking about this when I write my code... but there's something bugging me. Looking at the deployment descriptor, it lets you say who can access which bean type. But you can't say which *instance* of that bean type.
I know what he's getting at, because I was thinking the same thing. What do I do if I want employees to be able to have access to some of their own data—but only their OWN data. How do I set individual entity-level control? Otherwise, I'd have to either block ALL employees from these methods, or let ALL employees have access to the methods that would work on ANY entity bean. How can I let the role of 'employee' call, say, getSalary() but only see his OWN salary?

Class-level vs. instance-level security

When declarative security is not enough, you might need programmatic security to restrict access to a specific instance of a bean.

So far, all the security we've looked at has been *declarative*...not hard-coded into the bean class. Declarative security is cool because it supports the whole idea of component-based development—you can customize the bean at deploy-time without touching the code. Company A might be using a bean in one application, and need a particular type of access control that's completely different from the way Company B is using that same bean. Or even two uses of the same bean in the same company might need different access control.

But declarative security is at the class-level. You specify which methods a particular role can call, but it means that role can call the method on ANY instance of the bean class. If you need instance-level security, you can't do it in the deployment descriptor. But you can do programmatic security, which of course you already know... you've seen the two security-related methods in SessionContext and EntityContext.

```
public void doSecurity() {
   java.security.Principal p = context.getCallerPrincipal();
   String name = p.getName();
   // now do a comparison by checking the name
   // against the persistent field in the bean
   // that should match the principal name
}
```

But be careful! There is no guarantee that the name coming from the getName() method on the principal matches the user's log-in name. So you have to assume that as soon as you put programmatic security into your bean, you're drastically reducing it's reusability, and portability. And it means the Bean Provider might need to be more tightly-coupled to the operational environment than you'd normally want. Because the Bean Provider is making an assumption that the name coming from the getName() method will match a persistent field in the entity bean, and that might not be true in all environments, depending on how your system handles security.

Using programmatic security to custom-tailor a method

We just looked at getCallerPrincipal(), to see how you could find out exactly WHO the container believes is calling the method. But we can also use the isCallerInRole() method to test whether the principal (*whoever* it is and in this case we don't care) is a member of a specific *role* that we do care about. Imagine that you have a private pricing method, called by one of your business methods, that checks to see if this customer authenticated as a member of your special VIP customer group. If he is, you treat him differently than if he isn't. Maybe you give him a discount. Maybe you raise the price. Whatever your business logic tells you to do.

```
  private void determinePrice() {

      if (context.isCallerInRole("VIPCustomer")) {
         // treat customer well
      } else {
        // treat him like the loser he is
      }
   }
```

With isCallerInRole(), you can use security information to tailor the way a method runs, depending on which role called the method.

This is usually NOT a good way to handle security. To control access to a method, you're much better off using the method permissions to stop an unauthorized role from ever calling the method.

But you can still USE the caller's role information to do other non-security things in your code.

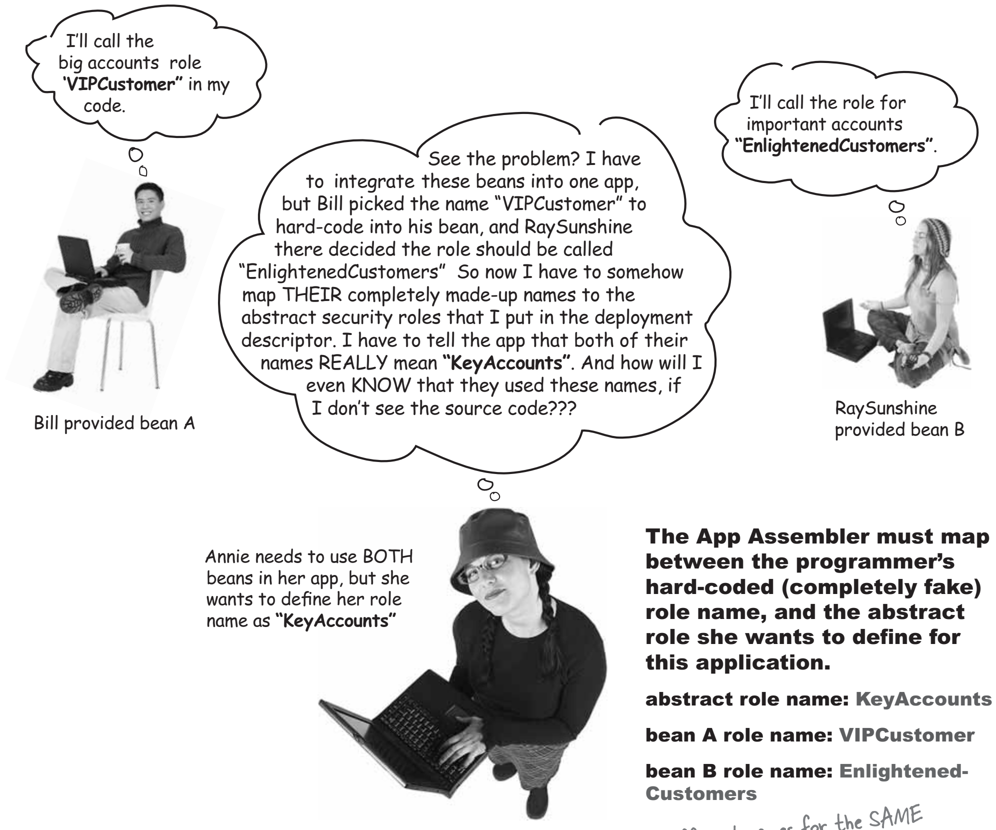
Besides the obvious problem with hard-coding security information (cuts down your reuse, increases the chances that things won't work in a portable way, etc.), there's another big problem—how the heck does the Bean Provider know in advance what roles the App Assembler is going to assign?

He doesn't. Or at least, he probably shouldn't. We know, we know. In reality, the Bean Provider and App Assembler are, if not the same person, closely related (professionally-speaking, of course). But we're still pushing for a component-based development model here, and at the very least, we want the App Assembler to take what the Bean Provider has done and somehow fit it into her own application. And that might mean integrating the bean with beans from other providers.

So now imagine this scenario: Annie the App Assembler is building a new app from four existing beans, two of which came from different providers. Providers who didn't work for the same company or know one another. *How could they have communicated in advance*, to know which roles Annie was going to choose for her application?

The problem with isCallerInRole()...

the Bean Provider hard codes a role name, but how does he know what roles the App Assembler will use? And what if there's more than one Bean Provider?


I'll call the big accounts role **"VIPCustomer"** in my code.
See the problem? I have to integrate these beans into one app, but Bill picked the name "VIPCustomer" to hard-code into his bean, and RaySunshine there decided the role should be called "EnlightenedCustomers" So now I have to somehow map THEIR completely made-up names to the abstract security roles that I put in the deployment descriptor. I have to tell the app that both of their names REALLY mean **"KeyAccounts"**. And how will I even KNOW that they used these names, if I don't see the source code???
I'll call the role for important accounts **"EnlightenedCustomers"**.
Bill provided bean A
RaySunshine provided bean B
Annie needs to use BOTH beans in her app, but she wants to define her role name as **"KeyAccounts"**

The App Assembler must map between the programmer's hard-coded (completely fake) role name, and the abstract role she wants to define for this application.

abstract role name: KeyAccounts

bean A role name: VIPCustomer

bean B role name: Enlightened-Customers

three different names for the SAME role!! Annie wants the deployer to map REAL customer groups to "KeyAccounts" and not those other two names, but those names are in code, and she can't touch the Java source code!

Map declarative security roles to the programmer's hard-coded (fake) roles

The App Assembler maps between security role *references* (hard-coded role names the programmer made up) and security *roles* (abstract role names she chose for this app) using the **`<role-link>`** element.

Bean Provider

Oh yeah... the App Assembler needs to know that I put a role name in my code. I'll make a security role ***reference*** in the DD I make for this bean...

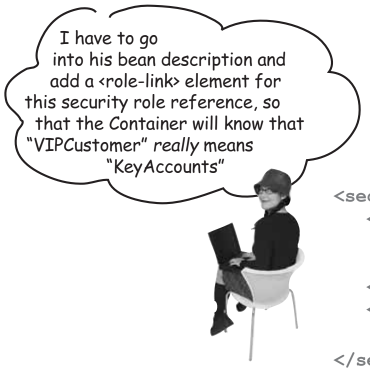
The Bean Provider has to TELL the App Assembler what he did in code, by putting in a <security-role-ref> tag

```
<enterprise-beans>
   <session>
      <ejb-name>ShoppingCart</ejb-name>
      ...
      <security-role-ref>
         <description>
            this role should be assigned to the
            accounts that get special VIP pricing
         </description>
         <role-name>VIPCustomer</role-name>
      </security-role-ref>
     ...
    </session>
</enterprise-beans>
```

These names must MATCH. You don't put in the 'real' role name, you put in the one you used as the argument to isCallerInRole()

```
context.isCallerInRole("VIPCustomer")
```

App Assembler

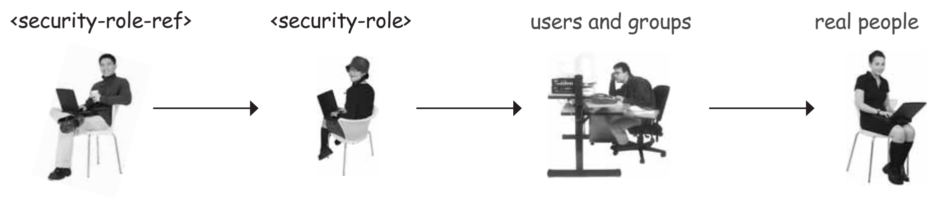

I have to go into his bean description and add a <role-link> element for this security role reference, so that the Container will know that "VIPCustomer" *really* means "KeyAccounts"

The App Assembler adds a <role-link> to what the Bean Provider put in his <security-role-ref>, to map the CODED role to the App Assembler's true abstract role

```
<security-role-ref>
   <description>
      this role should be assigned to the
      accounts that get special VIP pricing
   </description>
   <role-name>VIPCustomers</role-name>
    <role-link>KeyAccounts</role-link>
</security-role-ref>
```

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The complete security mapping picture

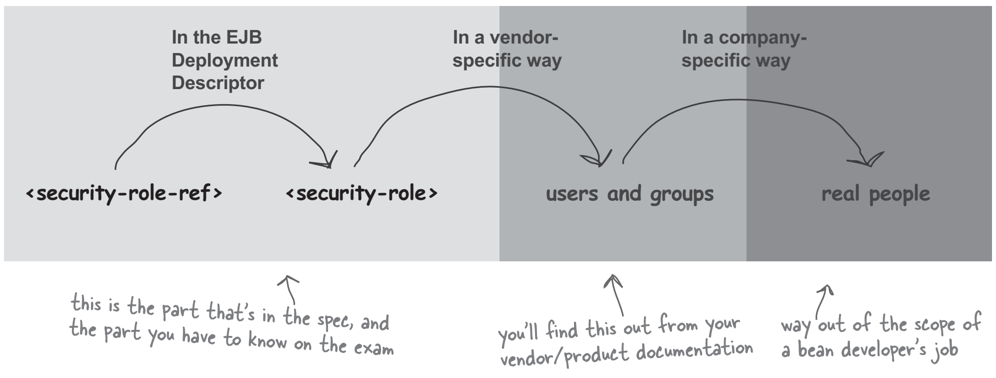


App Assembler maps the programmer's hard-coded security role references to her abstract security roles, using the <role-link> element.

Deployer maps the App Assembler's abstract security roles to users and groups in the Deployer's company, using a vendor-specific mapping (could be a properties file).

Someone in the operational environment sets up users and groups in the real company. Probably some kind of sys admin responsible for assigning log-in info and passwords to employees.

Where and how the mapping happens

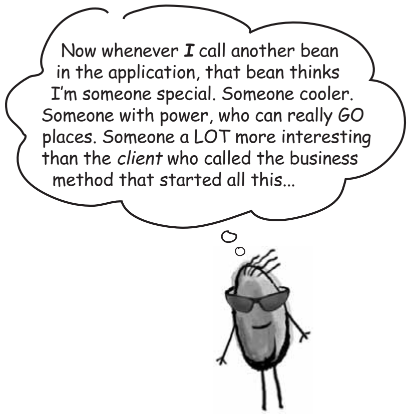

In the EJB Deployment Descriptor
In a vendor-specific way
In a company-specific way
<security-role-ref>
<security-role>
users and groups
real people
this is the part that's in the spec, and the part you have to know on the exam
you'll find this out from your vendor/product documentation
way out of the scope of a bean developer's job

Use <run-as> security identity to pretend someone else is calling...

When a client calls a method, the Container always knows the client's Principal, which includes the abstract security roles the Deployer assigned to that Principal. And remember, the caller's security context is propagated throughout the application as the client's original method goes about doing its work. Each method called in the conceptual call stack will get the security context along with the call.

But... let's say that you don't want the client's security context to keep propagating. Let's say that when the client calls Bean A, and Bean A in turn calls Bean B, you want Bean B to think that someone else is calling. In other words, what if you want Bean A to pretend to be someone else? That way, any bean that Bean A calls will think the Principal (and roles) of the caller is something other than the original client's.

Why would you do this? Bean B might have tighter access control. Perhaps Bean B won't allow outside clients to call its methods, so it doesn't have method permissions set up for any of the abstract roles mapped to clients users and groups. But perhaps you have a special role set-up just for other beans, and Bean B will take calls from other beans, as long as those beans are in that role.


Now whenever *I* call another bean in the application, that bean thinks I'm someone special. Someone cooler. Someone with power, who can really GO places. Someone a LOT more interesting than the *client* who called the business method that started all this...

When you want the bean to BE someone other than the calling client

```
<enterprise-beans>
   <session>
       <ejb-name>BeanA</ejb-name>
       ...
       <security-identity>
          <run-as>
             <role-name>SuperBean</role-name>
          </run-as>
       </security-identity>
       ...
    </session>
</enterprise-beans>
```

role-name must match a real abstract security-role set up by the App Assembler

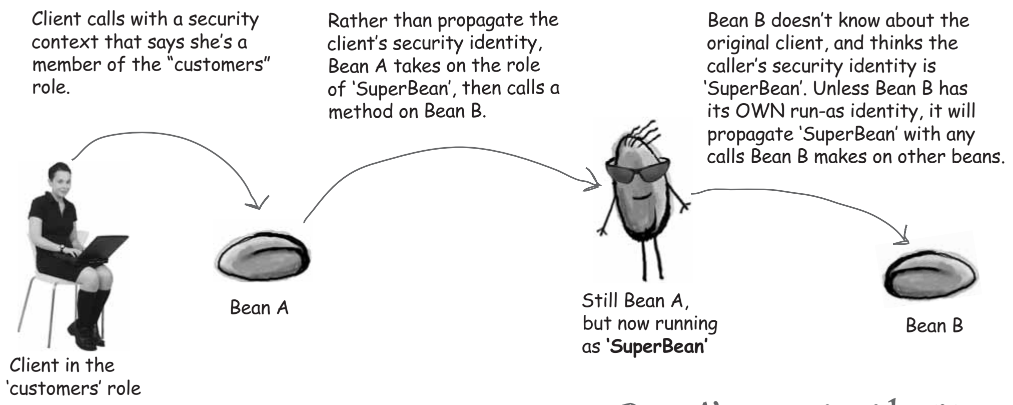
Explicitly saying that you want the calling client's identity to be used

```
<enterprise-beans>
   <session>
       <ejb-name>BeanA</ejb-name>
       ...
       <security-identity>
          <use-caller-identity/>
       </security-identity>
       ...
    </session>
</enterprise-beans>
```

This is what you get by default, if you don't put in a <security-identity> element at all. So you don't have to say this, but you can.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Security context propagation with <run-as>


Client calls with a security context that says she's a member of the "customers" role.
Rather than propagate the client's security identity, Bean A takes on the role of 'SuperBean', then calls a method on Bean B.
Bean B doesn't know about the original client, and thinks the caller's security identity is 'SuperBean'. Unless Bean B has its OWN run-as identity, it will propagate 'SuperBean' with any calls Bean B makes on other beans.
Client in the 'customers' role
Bean A
Still Bean A, but now running as **'SuperBean'**
Bean B

Message-driven beans must NOT say <use-caller-identity/>!!

Which caller would that be? Message-driven beans don't HAVE a calling client! So watch out for a deployment descriptor that shows a message-driven bean with a security identity that says <use-caller-identity/>. However, message-driven beans CAN have a <run-as> identity. And while we're here, don't forget that a message-driven bean must never try to ask its MessageDrivenContext for the calling client's security identity. A <security-identity> doesn't specify who the bean thinks is calling IT, but rather who the bean pretends to be when it makes calls to OTHER beans.

Bean A's <security-identity> is not about who Bean A should see as its caller, but rather what Bean B will see when Bean A calls it.

In other words, <security-identity> is not about changing the incoming client identity, but about changing what the bean propagates as the outgoing identity, when the bean calls other beans.

(Like registering at a hotel under the name "Jennifer Lopez", when you're ACTUALLY "Simon Roberts"... and the hotel clerk can't tell. As far as he is concerned, you're J-Lo.)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

COFFEE CRAM

Mock Exam

1 What's true about security for EJBs? (Choose all that apply.)

❑ A. All security policies must be expressed declaratively.

❑ B. The default security principal under which a method invocation is performed is that of the component's creator.

❑ C. Using EJBs, method permissions can be declared using EJB QL in the deployment descriptor.

❑ D. Security authorization can be bypassed on a method by method basis.

❑ E. Security authorization can be bypassed on an instance by instance basis.

2 What's true about methods that should run without being checked for authorization? (Choose all that apply.)

❑ A. They can be listed in the <exclude-list> element.

❑ B. They can be listed in the <unchecked> element.

❑ C. When the <unchecked> element is used, it should be placed where the <role-name> element normally occurs in the deployment descriptor.

❑ D. When a method permission relation specifies both <unchecked> and a security role, the container will use the security role.

3 Which role(s) should typically define the appropriate security policies for an application? (Choose all that apply.)

❑ A. bean provider

❑ B. application assembler

❑ C. deployer

❑ D. system administrator

❑ E. server provider

4 In which of the following methods can a stateful session bean invoke the **`isCallerInRole`** method in order to perform a security check? (Choose all that apply.)

- ❑ A. **`ejbCreate()`**
- ❑ B. **`ejbActivate()`**
- ❑ C. **`ejbPassivate()`**
- ❑ D. None of the above

5 Which are the bean provider's responsibilities when making an application secure? (Choose all that apply.)

- ❑ A. Assigning the security domain to the application.
- ❑ B. Authentication of principals.
- ❑ C. Mapping the principals used by the client to the principals defined for the bean.
- ❑ D. None of the above.

6 In which of the following methods can a CMP entity bean invoke the getCallerPrincipal() method in order to perform a security check? (Choose all that apply.)

- ❑ A. **`ejbCreate()`**
- ❑ B. **`ejbActivate()`**
- ❑ C. **`ejbPassivate()`**
- ❑ D. **`ejbPostCreate()`**
- ❑ E. Business methods

7 In which of the following methods can a BMP entity bean invoke the **`getCallerPrincipal()`** method in order to perform a security check? (Choose all that apply.)

- ❑ A. **`ejbLoad()`**
- ❑ B. **`ejbStore()`**
- ❑ C. **`ejbPassivate()`**
- ❑ D. **`ejbPostCreate()`**
- ❑ E. Business methods

8 When describing method permissions in the deployment descriptor for a specific bean, what's true? (Choose all that apply.)

- ❑ A. A wild-card character can refer to all of the bean's methods.
- ❑ B. Individual overloaded methods cannot be distinguished from each other.
- ❑ C. A method in the home interface cannot be distinguished from a method with the same name in the component interface.
- ❑ D. Individual methods can be referred to.

9 The **`<role-name>`** element can be used within what other security related deployment descriptor element(s)? (Choose all that apply.)

- ❑ A. **`<security-role>`**
- ❑ B. **`<run-as>`**
- ❑ C. **`<method-name>`**
- ❑ D. **`<exclude-list>`**
- ❑ E. **`<security-role-ref>`**

10 Which roles have what responsibilities when implementing security for EJB applications? (Choose all that apply.)

- ❑ A. The Application Assembler typically specifies when run-as identity should be used in an application.
- ❑ B. The Bean Provider maps security role references to security roles.
- ❑ C. The Bean Provider is typically responsible for assigning the security domain and principal realm to the application.
- ❑ D. The Deployer maps security role references to security roles.

11 Within the **`<security-role-ref>`** deployment descriptor element, which sub-elements are optional? (Choose all that apply.)

- ❑ A. **`<role-name>`**
- ❑ B. **`<role-link>`**
- ❑ C. **`<description>`**
- ❑ D. None of the above are optional

COFFEE CRAM

Mock Exam Answers

1 What's true about security for EJBs? (Choose all that apply.) (spec: 434-435)

❏ A. All security policies must be expressed declaratively.

❏ B. The default security principal under which a method invocation is performed is that of the component's creator.

❏ C. Using EJBs, method permissions can be declared using EJB QL in the deployment descriptor.

[✓ D. Security authorization can be bypassed on a method by method basis.]

❏ E. Security authorization can be bypassed on an instance by instance basis.

2 What's true about methods that should run without being checked for authorization? (Choose all that apply.) (spec: 443)

❏ A. They can be listed in the <exclude-list> element. – For methods that must NEVER be called

[✓ B. They can be listed in the <unchecked> element.]

[✓ C. When the <unchecked> element is used, it should be placed where the <role-name> element normally occurs in the deployment descriptor.]

❏ D. When a method permission relation specifies both <unchecked> and a security role, the container will use the security role.

3 Which role(s) should typically define the appropriate security policies for an application? (Choose all that apply.) (spec: 435-436)

❏ A. bean provider

[✓ B. application assembler]

[✓ C. deployer]

❏ D. system administrator

❏ E. server provider

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

4 In which of the following methods can a stateful session bean invoke the **`isCallerInRole`** method in order to perform a security check? (Choose all that apply.) (spec: 80)

- [✓] A. **`ejbCreate()`**
- [✓] B. **`ejbActivate()`**
- [✓] C. **`ejbPassivate()`**
- ❑ D. None of the above

5 Which are the bean provider's responsibilities when making an application secure? (Choose all that apply.) (spec: 448–454)

- ❑ A. Assigning the security domain to the application. Typically the Deployer's job
- ❑ B. Authentication of principals.
- ❑ C. Mapping the principals used by the client to the principals defined for the bean.
- [✓] D. None of the above.

6 In which of the following methods can a CMP entity bean invoke the getCallerPrincipal() method in order to perform a security check? (Choose all that apply.) (spec: 179–180)

- [✓] A. **`ejbCreate()`**
- ❑ B. **`ejbActivate()`** – who would the client be?
- ❑ C. **`ejbPassivate()`**
- [✓] D. **`ejbPostCreate()`**
- [✓] E. Business methods

7 In which of the following methods can a BMP entity bean invoke the **`getCallerPrincipal()`** method in order to perform a security check? (Choose all that apply.) (spec: 257)

- [✓] A. **`ejbLoad()`**
- [✓] B. **`ejbStore()`**
- ❑ C. **`ejbPassivate()`** – who would the client be?
- [✓] D. **`ejbPostCreate()`**
- [✓] E. Business methods

8 When describing method permissions in the deployment descriptor for a specific bean, what's true? (Choose all that apply.) (spec: 444)

- ☑ A. A wild-card character can refer to all of the bean's methods.
- ❑ B. Individual overloaded methods cannot be distinguished from each other.
- ❑ C. A method in the home interface cannot be distinguished from a method with the same name in the component interface. The <method-intf> element does this
- ☑ D. Individual methods can be referred to.

9 The **`<role-name>`** element can be used within what other security related deployment descriptor element(s)? (Choose all that apply.) (spec: 444-447)

- ☑ A. **`<security-role>`**
- ☑ B. **`<run-as>`**
- ❑ C. **`<method-name>`**
- ❑ D. **`<exclude-list>`**
- ☑ E. **`<security-role-ref>`**

10 Which roles have what responsibilities when implementing security for EJB applications? (Choose all that apply.) (spec: 446-449)

- ☑ A. The Application Assembler typically specifies when run-as identity should be used in an application.
- ❑ B. The Bean Provider maps security role references to security roles. Usually defined by the Application assembler
- ❑ C. The Bean Provider is typically responsible for assigning the security domain and principal realm to the application.
- ❑ D. The Deployer maps security role references to security roles.

11 Within the **`<security-role-ref>`** deployment descriptor element, which sub-elements are optional? (Choose all that apply.) (spec: 527)

- ❑ A. **`<role-name>`**
- ☑ B. **`<role-link>`** Usually defined by the Application assembler
- ☑ C. **`<description>`**
- ❑ D. None of the above are optional