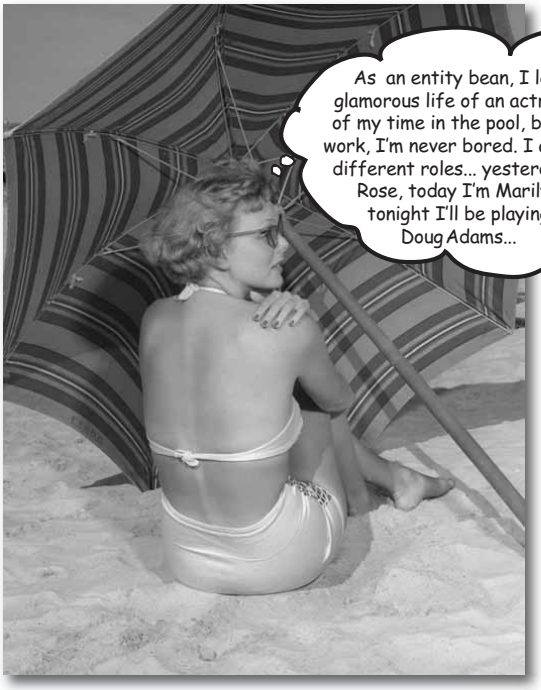
Table of Contents

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

6 bean/entity synchronization

Being an Entity Bean



Entity beans are actors. As long as they're alive, they're either in the pool or they're *being* somebody. Somebody from the underlying persistent store. In other words, an entity from the database. When a bean is playing a part, the bean and the underlying entity have to stay in sync. Imagine the horror if the bean is pretending to be, say, Audrey Leone, and someone lowers Audrey's credit limit in the database... but forgets to tell the bean. The bean, acting as Audrey, is happily authorizing purchases for more than Audrey's current limit. Or what if a client uses the bean to modify Audrey's address, but the bean hangs on to the new info without telling the database... yikes!

this is a new chapter 295

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

exam objectives

Entity Bean Lifecycle

Official:		What it really means:
6.1	Identify correct and incorrect statements or examples about the Bean Provider's view and programming contract for CMP, including the requirements for a CMP entity bean.	This objective can hit you on almost *anything* related to entity beans, so you pretty much have to know it all, including the details of a CMP entity bean lifecycle, the container callback methods of javax.ejbEntityBean, what you must write in a bean class, and what a bean can get from its EJBContext.
6.6	Identify the interface(s) and methods a CMP entity bean must and must not implement.	You have to know that passivation in entity beans is different from session bean passivation, because entity beans go back to a pool after ejbPassivate(), but stay as heal-living objects. You must know that entity bean creation is also completely different for entity beans, and that a create() call on an entity bean means "insert a new entity into the underlying persistent store." You have to know that remove() on an entity bean is more drastic for entity beans than session beans—an entity bean remove deletes the entity from the database!
7.1	Identify correct and incorrect statements or examples about the lifecycle of a CMP entity bean.	You must be able to look at an entity bean method and know the circumstances under which that method is called, and what you should do in that method. For example, you need to know that in a bean's ejbCreate() method, the bean cannot get a reference to its EJB object, but that it *can* get that reference in ejbPostCreate().
7.2	From a list, identify the purpose, behavior, and responsibilities of the Bean Provider for a CMP entity bean, including but not limited to: setEntityContext(), unsetEntityContext(), ejbCreate(), ejbPostCreate(), ejbActivate(), ejbPassivate(), ejbRemove(), ejbLoad(), ejbStore(), ejbFind(), ejbHome(), and ejbSelect().	You need to know which methods you're required to write into your bean, and which are left to the Container. For example, you need to know that the Bean Provider must write abstract getters and setters for persistent fields, but must define finder methods only in the home interface, not the bean class.
7.3	From a list, identify the purpose, behavior, and responsibilities of the Container for a CMP entity bean, including but not limited to: setEntityContext(), unsetEntityContext(), ejbCreate(), ejbPostCreate(), ejbActivate(), ejbPassivate(), ejbRemove(), ejbLoad(), ejbStore(), ejbFind(), ejbHome(), and ejbSelect().	

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

Entity Bean Lifecycle

Official:	*What it really means:*
8.1 From a list of behaviors, match them with the appropriate EntityContext method responsible for that behavior	You have to know the methods of EntityContext, as well as the circumstances under which you can call those methods. For example, you should know that if you're running in the setEntityContext() method, you can use your special JNDI namespace to look up a reference to a DataSource, but that you're not allowed to access the underlying database from that method (there's no meaningful transaction, so the Container won't let you do it.) You should know that you can't call getUserTransaction() on your context, because entity beans must use container-managed transactions, and getUserTransaction is for bean-managed transactions only.
8.2 Identify correct and incorrect statements about an entity bean's primary key and object identity.	You have to know that an entity bean must have a unique identity (no two beans can have the same primary key), and that a CMP bean's primary key must be either one of the bean's persistent fields, or a custom primary key class, whose fields are all from container-managed fields defined in the bean. You must know that the primary key class type must be Serializable, and that it must have a valid override of equals() and hashcode().

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

The real power of entity beans is synchronization

The bean and the underlying row in the database must stay in sync! Remember, the bean is not the entity—the bean is a *representation* of the real entity. The bean is an OO way of working with the data, but the entity in the database is the only true entity. If the entity in the database dies (i.e. is deleted), the bean for that specific entity dies too (although the bean *instance* still lives, as you saw in the last chapter).

The Container's job is make sure that the entity and the bean stay in sync, so that nobody's looking at a stale bean or a stale row in the database. Keep in mind that the entity *bean* may not be the only way to get into the database! In fact, in most cases the Bean Provider (you) will have no way to know for certain, during development, if your bean is the only way anyone will ever be able to get to the entity data.

For example, your Customer database might be used by a bunch of apps in your company, including some that work directly with the database and some that go through the Customer bean in an EJB application.

If a client has a reference to a bean, the client might change the bean's state by, say, calling a setter method. If that setter method corresponds to one of the bean's persistent fields (in other words, a column in the table like *address*), the bean and the database will be temporarily out of sync. The database won't have the current address until the bean updates the database!

Now think what a disaster it would be if the bean caches the new data for the entity, *without telling the database.* If someone comes along with another application and asks for that Customer's address from the database, That Would Be Bad.

And the opposite scenario is bad as well: if someone updates the database, ***the bean needs to know!*** Otherwise, the bean is out there in the EJB app, representing the entity in the database, but the bean isn't a true reflection of the entity's state. In other words, the bean is *stale.* Which means, perhaps, *useless.*

The Container always knows when the bean and the database (the underlying entity) must be synchronized, so that neither one has 'stale' data.

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

Oh no!

The entity *bean* and the *entity* it represents have different data!

Someone changed the bean state, but not the database!

First	Last	Key	Address	Phone
Chris	Martin	100	42 Foo st.	555-1212
Fran	Healy	16	200 Bar rd.	555-3434
Bela	Fleck	5	25 Pick lane	555-5656

The Container's most important entity bean job is to make sure that this scenario—where the bean and the database are out of sync—doesn't cause any damage.

The Container has to make sure that:

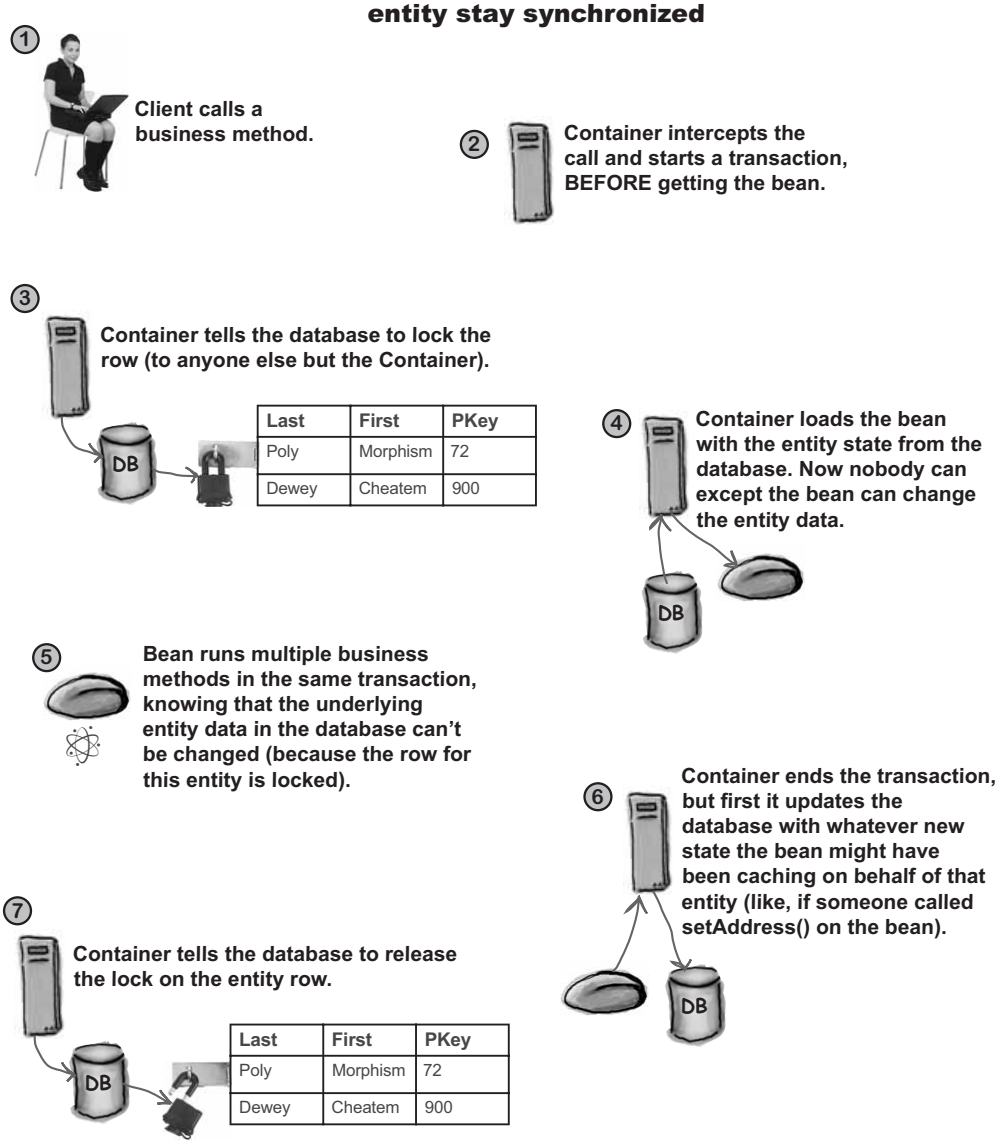
- While somebody is working with the bean (and potentially changing its state), nobody can work with the real entity in the database.
- Once an entity bean's state has been updated, the database has to be updated, before anyone else can access that record in the database.
- Before the bean can run any business methods on a particular entity's behalf, the bean has to be refreshed with the entity's state. In other words, before the entity bean for Joe Bloggs #88 can run a getCreditLimit() method, the Container has to load the entity bean up with the most current data for Joe Bloggs. Otherwise, the bean might return the wrong credit limit—the limit that was in place the last time the bean was loaded up with Joe's data from the database.

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

entity bean *synchronization*

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

How the entity bean and the underlying entity stay synchronized



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

The only question is WHO does the work when it's time to synchronize

The Container always knows *when* the bean and the database entity must be synchronized. It knows based on transactions. If a client calls a method on a bean, and that method starts a new transaction for the bean, the Container knows that the underlying entity in the database may have changed since the last time this bean was loaded up with this entity's data. So the Container forces the bean to refresh its state by loading in the entity's data.

And of course the reverse is true. If the bean completes a method, and this method was the end of the transaction, the Container will tell the database to release the lock on this entity in the database. ***But...*** during the time when the row was locked, the bean might have been happily caching data that represents the newly-changed state of the entity (like a new address or phone number). In other words, the client might have been calling setter methods on the bean, with the intention of causing an update in the real database. The client wants to update a record. So, the Container knows it must force the bean to update the data in the database with the bean's state, before the Container tells the database to release the lock on that entity.

OK, so the *when* is not the issue. And that's important, because otherwise you, the Bean Provider, would have to work out the business logic to know exactly when there was a danger that the bean and entity are out of sync. And you can just imagine how quickly you could corrupt the state of your database...

The real issue is *who* does the database access? And there are only two choices: it's either *you* or *the Container*.

If you write the database access code, you write JDBC statements in the callback methods the Container calls when its time to kick you and say, "Hey – time to go to the database!" On the other hand, if the Container writes the database code, you save yourself a lot of coding time and effort, and you'll almost always get better performance.

Container-Managed Persistence (CMP) means the Container takes care of all the database access code for synchronization, including adding and deleting entities (records / rows in the database).

Bean-Managed Persistence (BMP) means YOU write the database access code (the JDBC statements), for when the Container tells you its time to go to the database.

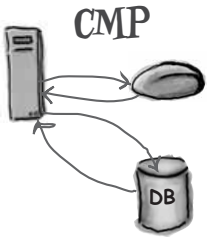
In EJB 2.0, you should use CMP. It saves you a lot of work, and virtually always gives you better performance.

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

Container-managed vs. bean-managed persistence

Container-managed persistence (CMP)

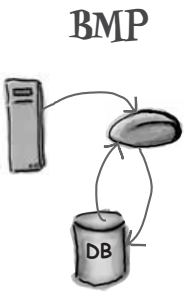
- Wimpy in EJB 1.1; greatly enhanced in EJB 2.0.
- The Bean Provider designs the entity bean class, choosing which of the bean's fields are part of the bean's persistent state. Persistent fields map to columns in one or more database tables.
- The Container keeps track of changes to the bean's state and updates the real entity as needed. For example, if a client calls setLastName() on the bean, the Container knows that the real entity in the database has to be updated, so that if anyone else accesses the database for that entity (including non-EJB clients using some other means to get to the database), they'll see the current state of the entity. The Container makes the decisions on how to keep the bean and the real entity synchronized based on the state of transactions. (More on that later).
- The Bean Provider writes EJB-QL to tell the Container how to do selects. EJB-QL is like a subset of SQL, but with some OO features. EJB-QL helps bridge the OO world of your bean to the relational world of your real entity.
- Using information in the deployment descriptor, the Container writes the actual implementation of the CMP bean, including implementing the finder and select methods, and *all* of the database access code. In other words, if you're a Bean Provider writing a CMP bean, you do *not* look up a resource connection factory for the DataSource, get an SQL Connection, or write any JDBC code. Not only is the data access code taken care of by the Container, but you can trust that the Container knows exactly when to go to the database.



The Container goes to the database when it needs to insert a new entity, delete and entity, update the database with new entity state (i.e. one or more columns in the entity row have changed, through the bean).

Bean-managed persistence (BMP)

- Bean Provider writes the database access code, including looking up a DataSource, getting a Connection, and sending JDBC statements to the database. It's still better than if you didn't use entity beans at all, because the Container will at least tell a BMP bean when to go to the database, so with BMP, you don't have to put in logic to keep the bean and the database in sync. When the Container tells you to update the database, you just do it.



The Container invokes a container call-back on the bean when the bean needs to do something with the database, and the bean code does the actual JDBC work.

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

there are no Dumb Questions

Q: Doesn't the Container still have to give you the database Connection? Isn't that the whole point of looking up a DataSource using resource factory references? If that's true, then how can you do bean-managed persistence? Or is BMP a way to bypass all that, in which case you'd be bypassing the Container services for connection pooling and...

A: Relax. Really. With BMP you still get your database connection from the Container, by looking up, as you said, a resource factory reference, javax.sql.DataSource. (Gee, you must have been reading ahead to the chapter on the EJB environment). Because you're right, you need to get a connection from a connection pool managed by the Container. But once you get the connection, for a BMP bean, you're on your own for sending it statements to do INSERT, DELETE, UPDATE, SELECT operations.

And while we're here, remember that database access is not just for entity beans. ANY bean can go to the database as part of its business logic. Session beans and message-driven beans might find plenty of reasons to look something up or store something in a database. The difference between entity beans and the other two bean types is that entity beans exist only because there's something in a database. Something the bean represents. So entity beans are, by definition, tied to something in a persistent store.

Message-driven and session beans do not represent something in a database, although they may need to *use* something in a database as part of their business logic.

Q: It seems like performance would be better—not worse—with BMP. Doesn't BMP give you more control?

A: You have more *flexibility* with BMP, but *not* necessarily more control. And according to nearly all benchmarks against the major J2EE servers, *not* better performance. That might sound counterintuitive, but remember that the Container can do things that you can't do from within bean code. Things like ganging multiple calls to the database, using native code to get in an out of the database faster than you could, lazy loading, dirty detection (neither of which are guaranteed by the spec, of course), and dozens of other tricks.

We know it goes against the conventional wisdom that says if you want it done right, do it yourself, but keep in mind that the server vendors are competing for your business. They know performance matters to you. The 2.0 spec added—and changed—a lot of the CMP specification for the sole purpose of giving the vendors more room for optimizing CMP.

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

A brief history on the evolution of CMP 2.0

Beginning with version 1.1 of the EJB spec, you could use both container-managed persistence (CMP) and bean-managed persistence(BMP). At the highest level, the difference between CMP and BMP is about who writes the database access code.

With EJB 2.0, you can still use both, but there's now very little reason to ever use BMP. The original EJB 1.1 specification for CMP entity beans was, um, weak. Clunky. Inefficient. Limited. Not That Good. And although many vendors were able to overcome many of the problems with CMP 1.1, the solutions were outside the specification, so your choice as bean developer was to use standard CMP but keep your portability, or step outside the spec (reducing your portability), but get better performance and features.

When the EJB 2.0 spec was created, the team spent a great deal of time and effort talking to both end-user customers (bean developers) and container vendors, all of whom were very happy to describe the problems with CMP. In graphic detail, complete with suggestions for Sun on, "I'll tell you what you can DO with your CMP bean spec..."

The EJB spec team listened. And designed. And listened. And designed. And in the end, they came up with something awful. A solution that was an equal-opportunity pisser-offer. Vendors hated it. Developers hated it. People like us who had to explain the new technology to customers really hated it. So with the deadline upon them, the team scrapped much of what they'd done with CMP and came up with a much cleaner solution.

With the new EJB 2.0 spec, CMP is lightyears ahead of where it was in EJB 1.1 (or in the first, terrible version of the pre-release EJB 2.0 spec). And now, most developers using entity beans will use CMP. In fact, the exam doesn't cover BMP at all, since it's there more for legacy support or very special cases more than anything else.

Because of the heavy shift to CMP (for reasons we'll explore), we won't cover BMP in this book. So from this point forward, we're going to assume that we're talking about CMP. The differences between the lifecycle, and the developer's responsibility, for CMP vs. BMP are dramatic, so don't forget that everything we talk about now will be from the perspective of a CMP bean, even if we don't explicitly say that we're referring to CMP.

The exam doesn't cover the history, gossip, or scandals of the EJB specification.

Although it would have made the book much spicier if it did. Ah well... we'll just have to make do with the technical content.

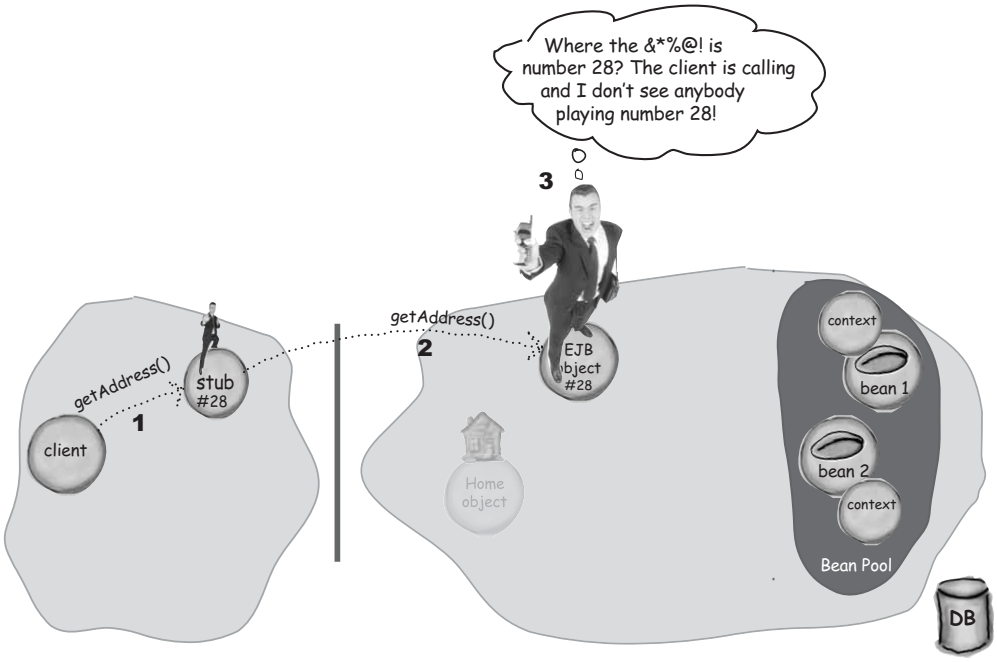
Still, we think the way a spec (any spec), evolves is interesting. Sun works hard to walk the fine line between having a meaningful spec with strong guarantees for the programmer, and one that's vague enough on implementation details to let the vendors compete on the performance of their implementations.

Regardless, there's nothing on the exam about the different versions of the spec. As long as you know what's in the 2.0 spec, you're safe. However, in the real world you're likely to come across EJB 1.1 applications, so be prepared to learn the differences. Especially if you're the lucky one in charge of migrating the app to a 2.0-compliant server ;)

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

How a bean *actor* becomes a bean *entity*

Scenario: client wants to get the address of a specific CMP Customer entity, primary key #28.



1 Client calls getAddress() on the stub for entity bean #28. The client got the stub from a previous call on the home stub (you'll see exactly how the client got the stub in the first place, in a minute).

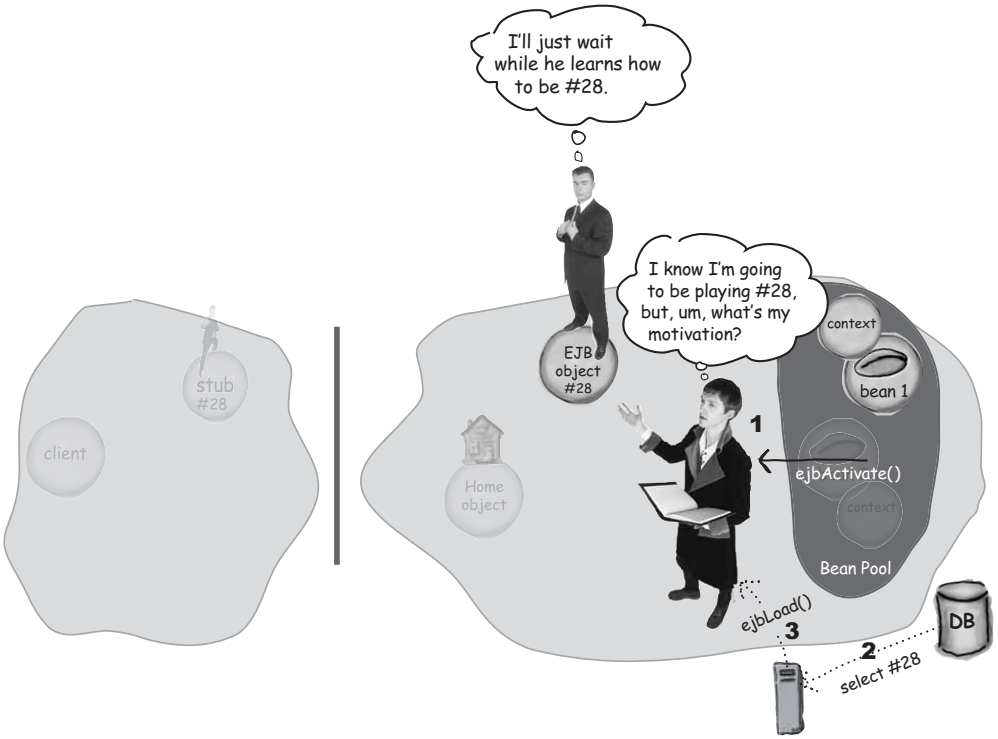
2 The call is passed to the EJB object.

3 The EJB object gets the call and panics, because there IS no entity bean for #28! The EJB object is the agent/bodyguard for #28, but he isn't attached to a specific bean that's playing #28.

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

How a bean *actor* becomes a bean *entity*

The CMP bean comes out of the pool and prepares to play #28.

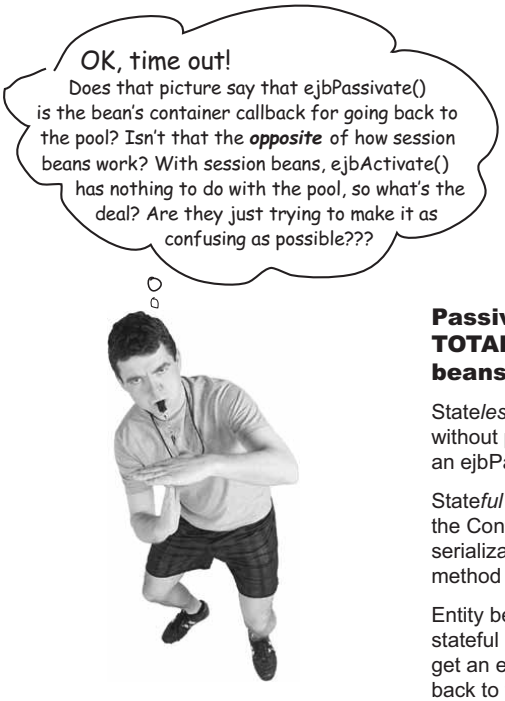


1 The Container 'activates' a bean by pulling it out of the pool and calling ejbActivate().

2 The Container does a select on primary key #28 in the database, to get the real entity data to put into the bean (the data that will become the bean's state).

3 The Container loads the bean with the entity data from the database, and then calls ejbLoad() to tell the bean, "Hey bean, you've just been loaded."

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



Passivation and Activation have a TOTALLY different meaning for entity beans!

State*less* session beans go back to the pool without passivation (in other words, without getting an ejbPassivate() call).

State*ful* session beans are passivated when the Container puts them to sleep (possibly serialization) to conserve resources between client method invocations.

Entity beans aren't passivated in the way that stateful session beans are, but entity beans DO get an ejbPassivate() call when they're about to go back to the pool (and an ejbActivate() when they come out of the pool).

The most important point is that, unlike session beans, ***passivated entity beans are still live objects on the heap!***

There's no such thing as a bean sleeping in a pool. Stateless beans, entity beans, and message-driven beans all use pools. And those pools are for living, RAM-using, on-the-heap objects.

Only state*ful* beans are put to sleep (called, confusingly, *passivation)*, but this has nothing to do with a pool.

So, yes, they've overloaded the word "passivation" just to make things really confusing for you and to help drive the need for more EJB books. (For which we thank them every single day.)

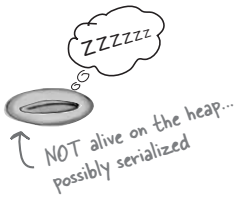
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

ejbPassivate(): the difference between session and entity beans

Entity bean

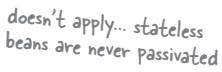
- Called when the bean has finished a business method (other than remove()) and is about to go back to the pool. In the passivated state, the bean has no identity—it's not representing any entity from the database!
- Use it to release resources that you don't want to waste while the bean is sitting in the pool, not running a business method. (Typically, the method is empty.)
- The bean is NOT passivated in the session bean sense. In other words, the bean is not serialized or saved, so there is NO requirement that you null out references to non-Serializable objects.
- The Container calls the bean's ejbActivate() method when a bean is needed to service a business method from a client. The bean runs ejbActivate() *before* it runs the business method that triggered the activation. (This method is usually empty as well).

State*ful* session bean



- Called when the Container decides to conserve resources between business method invocations from the client.
- Use it to release resources and to prepare the bean's state for what *might* be serialization (null out non-Serializable references, etc).
- The bean is "put to sleep" and is no longer taking up space on the heap.
- The Container calls the bean's ejbActivate() method when a client calls a business method on the EJB object. The bean runs ejbActivate() *before* it runs the business method that triggered the activation.

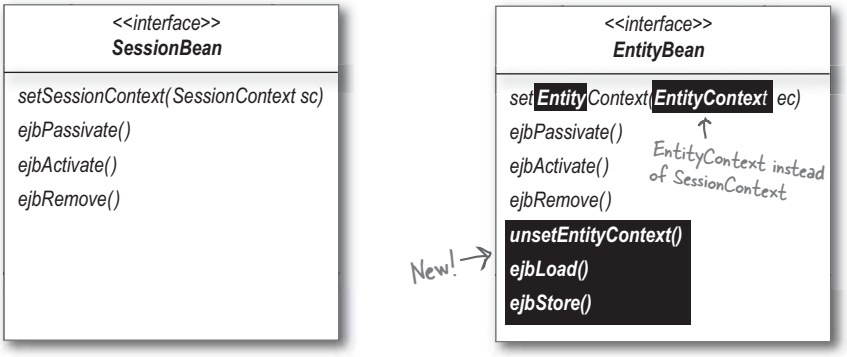
State*less* session bean



- Doesn't apply! Stateless beans have a pool, but activation and passivation don't play any part in it. Remember, stateless session beans will NEVER get an ejbActivate() or ejbPassivate() call.

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

The EntityBean interface adds three new container callbacks (including two just for synchronization)



The EntityBean interface adds three new methods (and the context setter changes to give the bean an EntityContext instead of a SessionContext). But... and this is an extremely large "but"... even the methods which are the same in both interfaces don't behave the same.

Of the four methods in SessionBean, only the context setter behaves the same as its counterpart in EntityBean. The other three, ejbPassivate(), ejbActivate(), and ejbRemove() have drastically different meanings.

You'll see.

For now, just be ready to let go of your attachments to the meaning of activation, passivation, and removal. What those mean to an entity bean is nothing like what they mean to a session bean.

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

Even the methods that are the same, don't behave the same

SessionBean interface

setSessionContext(SessionContext sc)

Container gives the bean a reference to its context.

ejbPassivate()

Called on a stateful session bean, when the bean is about to be serialized (or something like it).

ejbActivate()

Called on a stateful session bean, when the bean is reactivated following passivation (might be deserialization).

ejbRemove()

Called on a stateful bean when the client calls remove(). Called on a stateless bean when the Container wants to reduce the size of the pool.

Of the four methods that are in both SessionBean and EntityBean, only the context setter method behaves the same way! In an entity bean, activate, passivate, and remove are completely different from a session bean

EntityContext adds getPrimaryKey()

Both SessionContext and EntityContext extend EJBContext. EntityContext has only one method that SessionContext doesn't have—getPrimaryKey(). This is tricky, because there's a getPrimaryKey() in both the EJBObject and EJBLocalObject interfaces, that's exposed to BOTH session and entity bean clients (although a session bean client will get an exception for calling it). But the context version of getPrimaryKey() is exposed only to entity beans.

EntityBean interface

setEntityContext(EntityContext ec)

Container gives the bean a reference to its context.

ejbPassivate()

Called when the bean is about to return to the pool, following a transaction.

ejbActivate()

Called when the bean is taken out of the pool to service a client's business method call.

ejbRemove()

Called when the client calls remove(), and wants to delete this entity from the database!

unsetEntityContext()

Called when the Container wants to reduce the size of the pool.

ejbLoad()

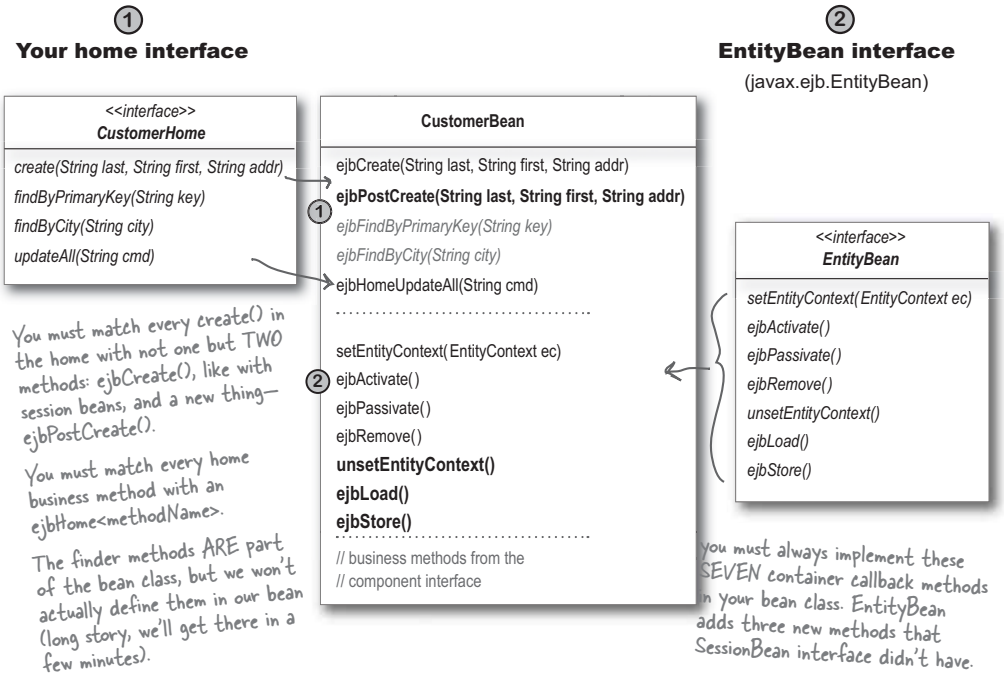
Called when the bean has been refreshed with data from the underlying persistent store.

ejbStore()

Called when the Container is about to update the database to reflect the state of the bean.

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

But wait... there's more! Entity beans have new home container callbacks, too

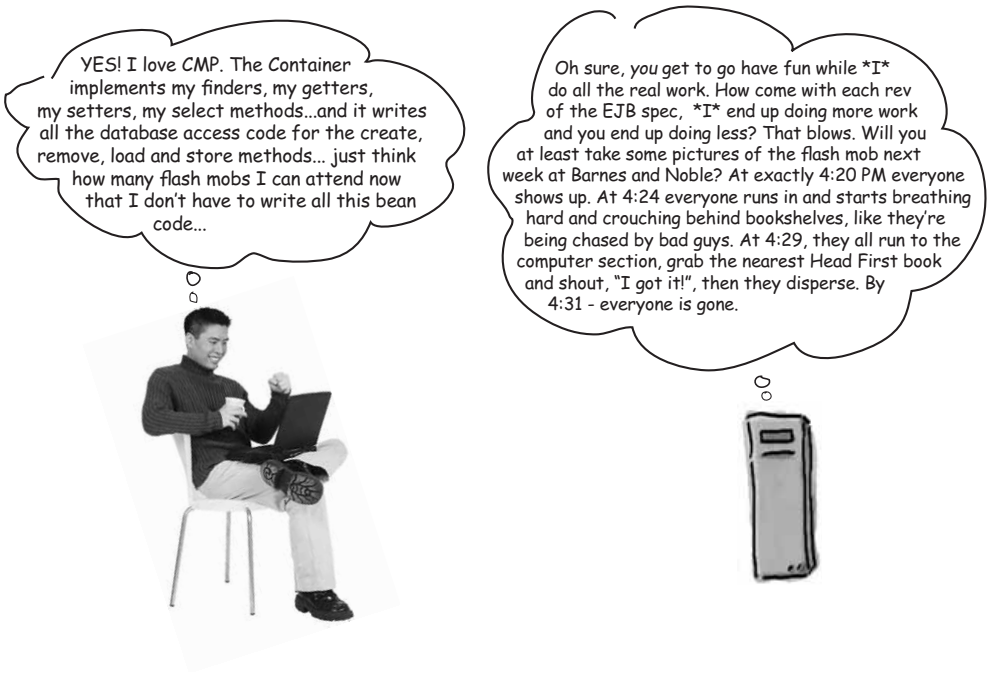


For any entity bean using container-managed persistence (CMP), you will always have at least seven container callbacks—all from the EntityBean interface implementation. You don't have to have a create() method in your home, but if you *do* have create() methods, you must match each create() with not one but *two* methods: ejbCreate() and ejb*Post*Create(). If you have home business methods, you must write a matching ejbHome<methodName> method in the bean class.

If you use bean-managed persistence (BMP), where you write your own database access code, you also must match each finder method with an ejbFind<whatever> method. Remember, the only method required in a home interface is findByPrimaryKey(). But in a CMP bean, you won't put any finders in your bean class even though they're in your home interface.

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

Writing a CMP entity bean: make it abstract



Make your CMP entity bean class abstract.

You still have to implement the container callbacks from javax.ejb.EntityBean, and if you still have to write all your business methods (including those from the home), and if you have any create methods, you have to match those in the bean class as well. But that still leaves a pile of work for the Container to do, including the accessor methods for your persistent fields. For example, if your Customer table has a column for address, and you map that to a field in your bean class (so that clients can, for instance, call getAddress()), you don't write the getters and setters for that field. In fact, ***you don't even declare the field!*** With CMP, you create a virtual field, by defining getters and setters. In other words, the *persistent field* for address exists in your bean *simply because there's an abstract getter and setter for it.* (Well... that's not *entirely* true. There also has to be an entry in the deployment descriptor, but we'll look at *that* in the next chapter).

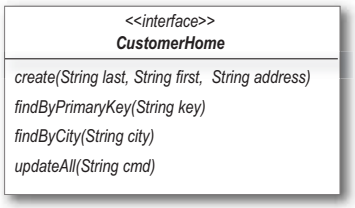
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

entity bean class

You put *three* kinds of things in your bean class:

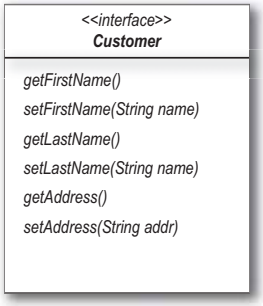
1 Things from the *home interface*

- For each create() method in the home, you must have a matching ejbCreate() and ejbPostCreate() method in the bean class.
- For each business method in the home, you must have a matching ejbHome<method> in the bean class.
- For CMP beans, you will NOT write matching finder methods. The Container will write them for you, based on info you put in the deployment descriptor. (We'll go over all that in the next chapter.)



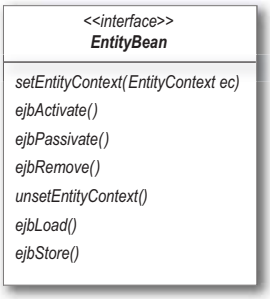
2 Things from the *component interface*

- For each method in the component interface, you must have a matching concrete implementation in the bean class.



3 Things from the *EntityBean interface*

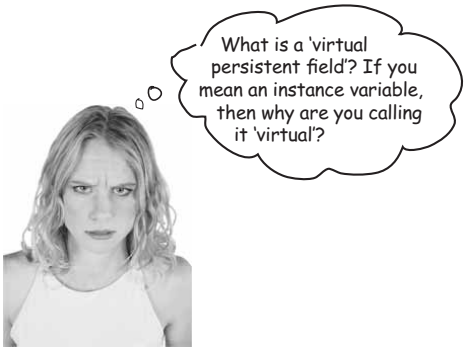
- You must implement the EntityBean interface either directly or indirectly. So unless you have a superclass that implemented the methods, you're responsible for writing concrete implementations in your bean class.



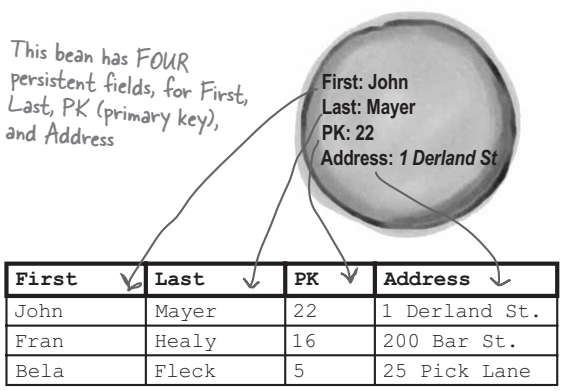
PLUS... (ok, that's *four* things...)

④ Virtual persistent fields

- For each persistent field, provide an abstract getter and setter



'Virtual persistent fields' are for the values that map to columns in the database. They represent the entity's persistent state. In your bean class code, they exist only as abstract getters and setters



First	Last	PK	Address
John	Mayer	22	1 Derland St.
Fran	Healy	16	200 Bar St.
Bela	Fleck	5	25 Pick Lane

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

Virtual fields are NOT instance variables!

```
public abstract class CustomerBeanCMP implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String last, String first, String addr) {
     this.setLast(last);
     this.setFirst(first);
     this.setPK(makePK());
     this.setAddress(addr);
     return null;
   }

   public abstract String getLast();
   public abstract void setLast(String last);
   public abstract String getFirst();
   public abstract void setFirst(String first);
   public abstract String getCustAddress();
   public abstract void setCustAddress(String addr);
   public abstract String getPK();
   public abstract void setPK(String pk);


   public String getLastName() {
      return this.getLast();
   }
   public void setLastName(String name) {
      this.setLast(name);
   }
   public String getFirstName() {
      return this.getFirst();
   }
   public void setFirstName(String name) {
      this.setFirst(name);
   }
   public String getAddress() {
      return this.getCustAddress();
   }
   public void setAddress(String addr) {
      this.setCustAddress(addr);
   }

   // more methods from EntityBean
   // and the home interface
}
```

note: the only instance variable is for the EntityContext!

These are the virtual persistent fields. They're nothing more than abstract getters and setters that map to the columns in the database, representing the entity's persistent state. The things that get saved with the entity!

These are methods from the component interface, that are exposed to the client.

We know what you're thinking... why do you have a SECOND set of getters and setters, when you can just expose the abstract ones in your component interface?

Well, you COULD expose them, but it's not a good idea. Think about it....

here's a hint: imagine that the non-abstract versions had more code than what you see here.

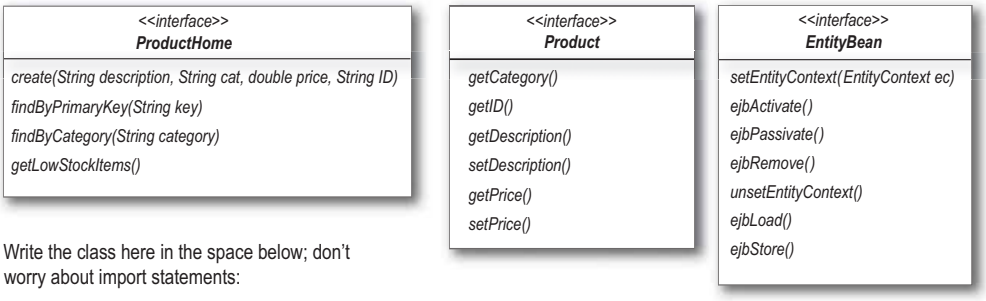
bigger hint: imagine that the non-abstract methods had VALIDATION code...

ridiculously big hint: think about encapsulation.

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

Sharpen your pencil

Using the interfaces below, write a legal bean class. You don't have to write the actual business logic, but at least list all the methods that you have to write in the class, with their correct declarations.



Write the class here in the space below; don't worry about import statements:

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

CustomerBeanCMP code

Complete code for the CustomerBeanCMP class

(note: it's not annotated, because that's YOUR job in the Sharpen exercise on the next page)

```
package headfirst;

import javax.ejb.*;

public abstract class CustomerBeanCMP implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String last, String first, String addr) {
     this.setLast(last);
     this.setFirst(first);
     this.setPK(makePK());
     this.setAddress(addr);
     return null;
   }

   public String getLastName() {
      return this.getLast();
   }

   public void setLastName(String name) {
      this.setLast(name);
   }

   public String getFirstName() {
      return this.getFirst();
   }

   public void setFirstName(String name) {
      this.setFirst(name);
   }

   public String getAddress() {
      return this.getCustAddress();
   }

   public void setAddress(String addr) {
      this.setCustAddress(addr);
   }

   public void setEntityContext(EntityContext ctx) {
      context = ctx;
   }
```


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

```
    public abstract String getLast();
    public abstract void setLast(String last);
    public abstract String getFirst();
    public abstract void setFirst(String first);
    public abstract String getCustAddress();
    public abstract void setCustAddress(String addr);
    public abstract String getPK();
    public abstract void setPK(String pk);

    public void unsetEntityContext() { }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }

    private String makePK() {
      int rand = (int) (Math.random() * 42);
      return ""+ rand;
    }
}
```

remember: this is the world's stupidest primary key generator. (Well, we suppose one that always returned the same number would be pretty lame, but that wouldn't work anyway, since primary keys MUST be unique or... guess what... the JVM will give you a nice DuplicateKeyException. But as we were saying, in the Real World, you'd either use a client-supplied arg as the primary key, or a REAL primary key generation engine of some sort, or ask the Container to give you the database auto-generated key (if your vendor supports that.)

Sharpen your pencil

1. Mark each method in the CustomerBeanCMP class with one of the following four symbols:

   H C EB VF

   based on the reason for that method's existence in the class. For example, the ejbCreate() method is required because there's a matching create() in the home, so mark an H next to the ejbCreate() method.

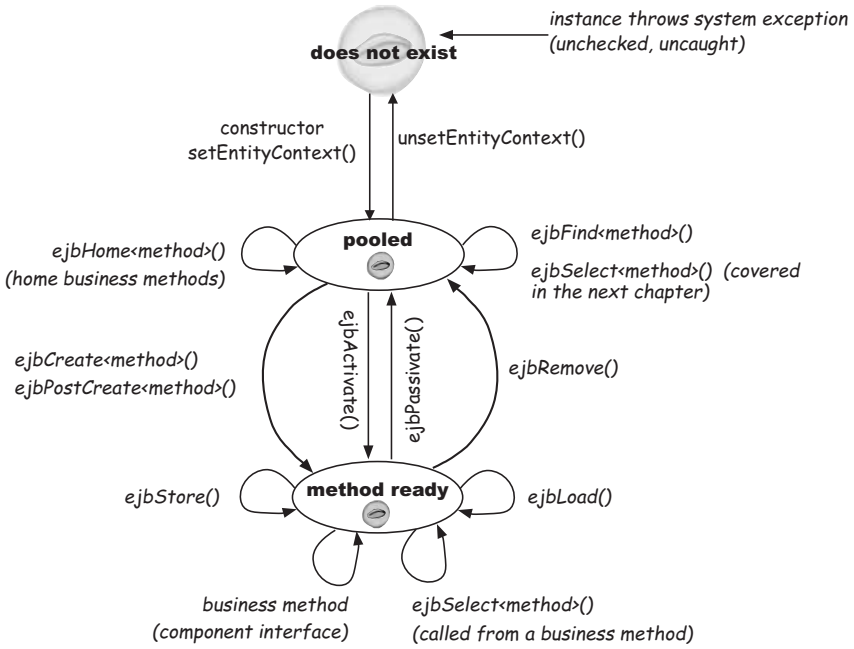
2. Put a check mark next to those methods that the compiler cares about. In other words, if you left a method out and the compiler would complain with an error, then mark that method with a ✓

3. Annotate the code yourself with any other details you can think of. For this exercise (but not the previous two), do as much as you can on your own, then turn back to earlier pages in this chapter and see if you can add or change anything.

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

entity bean lifecycle

Entity bean instance lifecycle



It looks bad only when you see the whole thing at once. Relax...by the end of this chapter, assuming you've regained consciousness, you'll have it 90% down, and by the end of chapter 7, you'll have it all.

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

Entity bean instance transitions

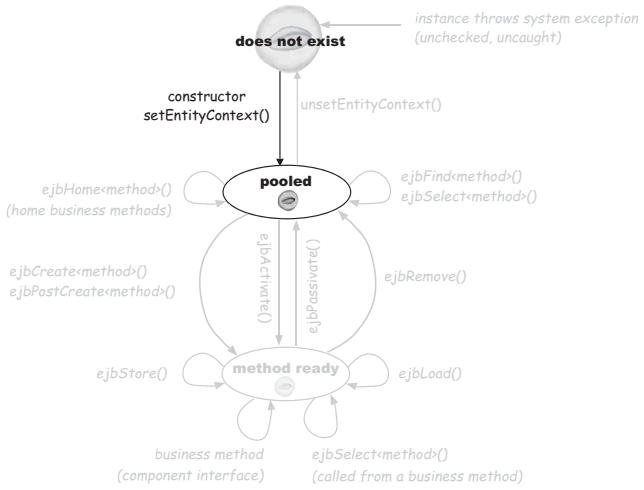
Don't try to memorize this all right now! We'll spend most of the rest of this chapter looking at this stuff.

Moving from *does not exist* to *pooled*

constructor

setEntityContext()

note: no create()!



Moving from *pooled* to *method-ready*

ejbCreate() / ejbPostCreate()

OR

ejbActivate()

(The Container does NOT call ejbActivate() if it calls ejbCreate())

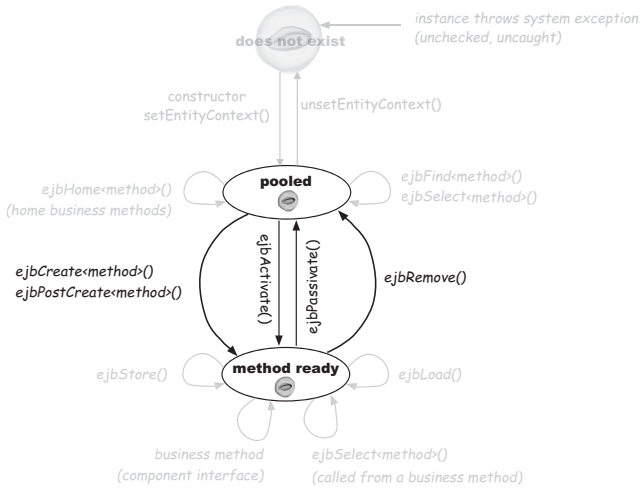
Moving from *method ready* to *pooled*

ejbPassivate()

OR

ejbRemove()

(Never both! A bean doing an ejbRemove() will NOT be passivated before going back to the pool!)



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

***entity state** transitions*

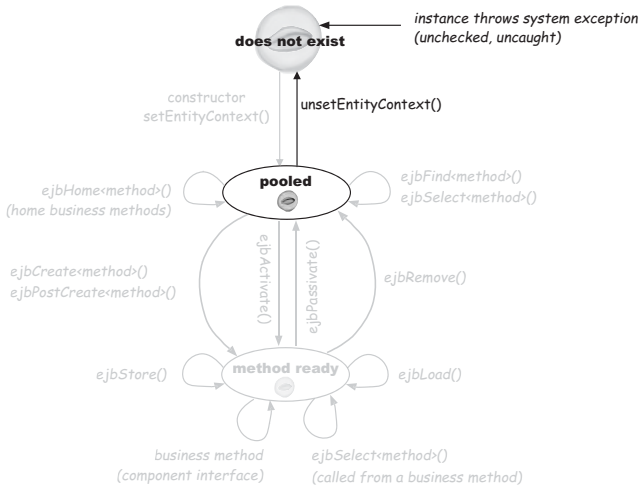
Entity bean instance transitions

Moving to *does not exist*

unsetEntityContext()

OR

instance throws a system exception



there are no Dumb Questions

Q: How come there is a label for ejbSelect<method > (whatever THAT is) in both the pooled state *and* the method-ready state. How can you run those methods in both?

A: We'll get to ejbSelect methods a bit later, but for now, think of them as private methods in the bean (in other words, called not by the client, but only by the bean's own methods) that do selects on the database. They're used only for CMP beans, and can be a huge convenience, since they're implemented for you by the Container. The reason they're in both places is because it depends on which interface the client uses to call the method that in turn calls a select method. So, if a client calls a home business method, and the home method calls an ejbSelect<method>, the bean stays in the pool to run the method. But if the select method is called from a method in the bean's component interface, that means the method is running on a specific entity, say, Frank Foof #56, so the bean is in the method-ready state.

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

there are no Dumb Questions

Q: It looks like there are TWO ways to move to the method-ready state: either the client calls a create() method or the Container calls ejbActivate(). So does this mean that you can't count on ejbActivate() being called each time you leave the pool?

A: That's right. A bean can move to the method-ready state by ONLY those two paths (creation or activation) but never both at the same time. So, if you have a design that acquires resources in ejbActivate(), so that they'll always be available while the bean is servicing a business method, you better grab them in ejbCreate() (or ejbPostCreate()—you'll see the difference in a few minutes).

In the real world, it's much less common in EJB 2.0 to use ejbActivate() for much of anything. We'll talk more about this both in this chapter and the last chapter (patterns and performance), but the short version is this: it's usually more efficient to acquire and release scarce resources just within the business methods that need them. That way, you're not hanging on to them (preventing other beans from having access) while your bean is active (i.e. not in the pool), but not actively running a method. Yes, that means you have some additional overhead in each business method, as opposed to grabbing the thing once in ejbActivate(), but in many cases the overhead of grabbing the resource is minor compared to the scalability cost of holding resources (we're thinking... database connections from the pool) open longer than you need to access those resources.

Bottom line: You'll probably find yourself leaving ejbActivate() empty, in so you won't have to worry about missing it when you come out of the pool via an ejbCreate() call.

Sharpen your pencil

For the exam, you have to know exactly which container callback methods are in the EntityBean interface, so you need to memorize these. The tricky part is that some of them have the same names, but completely different behavior than their session bean counterparts in the SessionBean interface. DO NOT LOOK ON THE OPPOSITE PAGE!

1. The client calls this method to tell the Container that he (the client) is done using the bean's EJB object reference:

_____

2. This method is called when the bean goes back to the pool, after an entity is deleted from the database:

_____

3. This method is called immediately after the bean's constructor runs:

_____

4. This method is called on the bean when the bean is in the pool, and the client invokes a business method on the component interface. (We're looking for the *first* method called in that scenario)

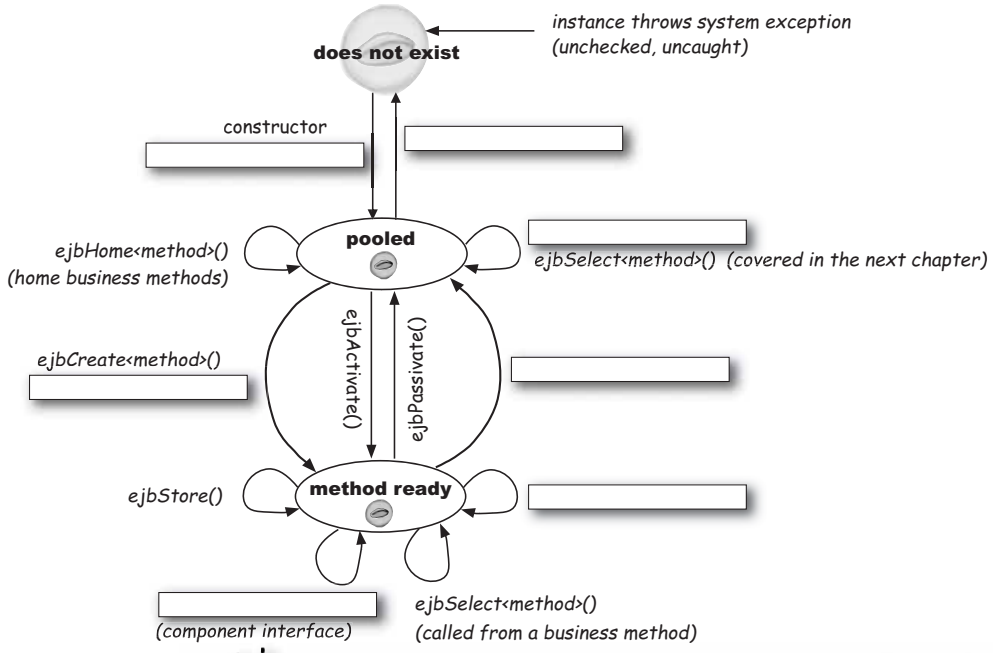
_____

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

***entity lifecycle** activity*

Sharpen your pencil

Fill in the missing methods. Don't worry if you don't get the name exactly right; just try to work out what happens at the transitions in the entity bean state diagram.



You have to know every detail of this lifecycle!

The exam expects you to know every detail of a bean's life. The trickiest parts, as we said earlier, are where the methods in an entity bean have the same names but entirely different meanings from those in a session bean. Take extra special care with:

***create** (make a new bean vs. make a new entity in the database)*
***remove** (tell the Container you're done vs. delete an entity in the database)*
***passivate** (serialize the stateful bean vs. send the bean back (whole) to the pool)*
***activate** (deserialize the stateful bean vs. bring it out of the pool to run a business method.)*
(remember, NO method is called when a stateless session bean goes in or out of the pool)

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

So how DID the client get a reference to the EJB object for #28?

When a client wants to call a business method on a specific entity (in other words, an entity with a unique primary key, like Lee Loo #28), the client first needs a reference to that entity's EJB object.

For an entity bean, remember, there are three ways that can happen:

1 Client calls a finder on the home

```
customerHome.findByPrimaryKey("28");
```

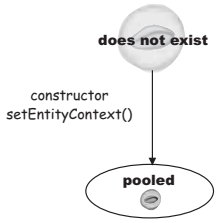
2 Client calls create on the home (to insert #28 for the first time)

```
customerHome.create("Loo", "Lee","28", "54 Bar Circle");
```

3 Client calls a home business method that returns a reference to the bean's component interface

```
customerHome.getCustomerByStreet("Bar Circle");
```

We'll talk about each stage of the bean's lifecycle for finders, creates, and home business methods. But first—each of these assumes that the bean instance already exists in the pool. But how does the bean get there in the first place?



Before the client can use a bean for ANYTHING -- creation, finders, business methods, etc., the Container has to make a new bean instance for the pool. Bean CONSTRUCTION isn't tied to entity bean CREATION.

Construction means a new bean instance. But creation, for an entity bean, means a new entity is inserted into the database.

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

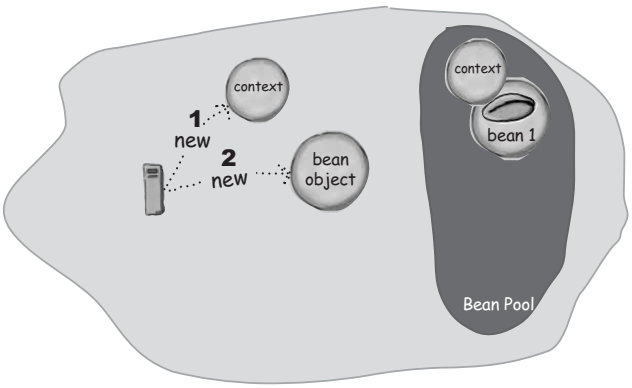
CMP bean construction

CMP Entity bean *construction*

Scenario: Container wants to make a new bean instance for the pool

1 Container makes a new EntityContext

2 Container makes a new instance of the bean class (bean's constructor runs)

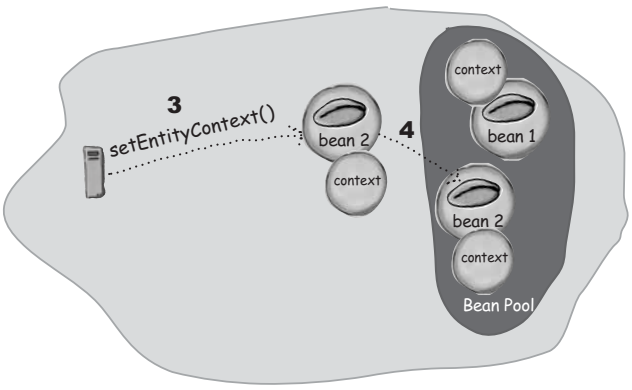


---

3 Container calls setEntityContext() on the bean. This is the ONLY time in a bean instance's life that it will get this call.

4 Container puts the bean (which now has a context) in the pool.

Notice: create() was never called! A create() method is *only* for inserting a new entity into the database, and has *nothing* to do with the bean instance's creation!



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

Bean things you can do during entity construction:

timeline

constructor

Use your EntityContext to:

- ☐ get a reference to your home
- ☐ get a reference to your EJB object
- ☐ get security information about the client
- ☐ force a transaction to rollback (CMT beans)
- ☐ find out if the transaction has already been set to rollback (CMT beans)
- ☐ get a transaction reference, and call methods on it (BMT beans)

You can't do ANYTHING

Access:

- ☐ your special JNDI environment
- ☐ another bean's methods
- ☐ a resource manager (like a database)

setEntityContext()

Use your EntityContext to:

- ☑ get a reference to your home
- ☐ get a reference to your EJB object
- ☐ get security information about the client
- ☐ force a transaction to rollback (CMT beans)
- ☐ find out if the transaction has already been set to rollback (CMT beans)
- ☐ get a transaction reference, and call methods on it (BMT beans)

Access:

- ☑ your special JNDI environment
- ☐ another bean's methods
- ☐ a resource manager (like a database)

What to put in the constructor

We know it's painfully obvious by now... NOTHING.

Unless you're forced to, don't even put a constructor in your code at all, and just use the compiler-generated default constructor.

Whatever you do, be SURE you have a public no-arg constructor in your class!

```
public Customer() { }
```

What to put in the setEntityContext() method

Assign the context to an instance variable. Remember, you get only ONE chance to save it. You might not always need to use a session context, but you'll probably need an entity context. Besides, your context is gonna stay alive whether you keep a reference to it or not, so the only memory you save if you don't keep it is for the reference variable, not the object itself.

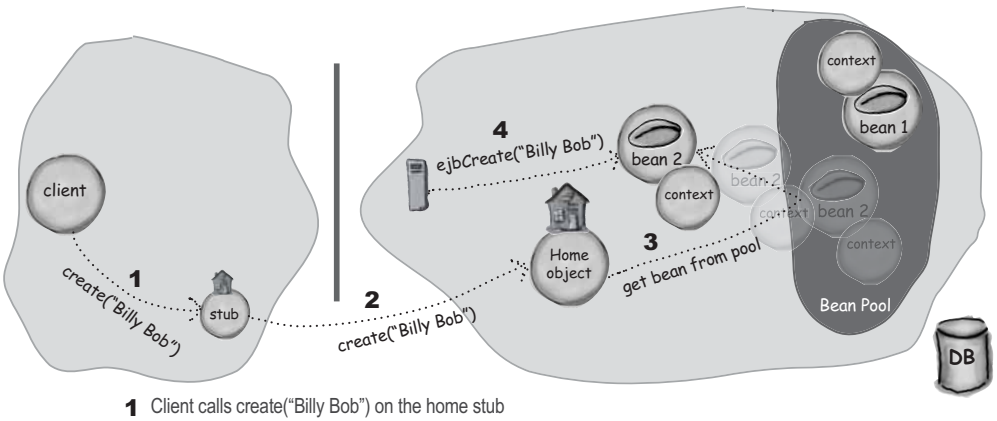
```
public void setEntityContext(EntityContext ctx) {
   context = ctx;
}
```


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

***CMP bean** creation*

CMP Entity bean creation

Scenario: client wants to create a new entity in the database (remember, the Container made the bean instance and the context earlier, and put them in the pool)

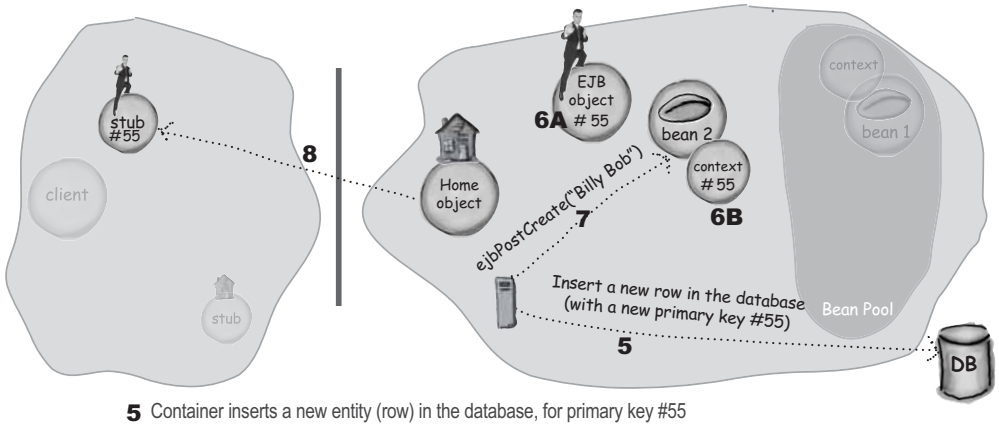


1 Client calls create("Billy Bob") on the home stub

2 The create("Billy Bob") method invocation is passed to the home object.

3 A bean is pulled out of the pool to do the creation

4 Container calls ejbCreate("Billy Bob") on the bean instance



5 Container inserts a new entity (row) in the database, for primary key #55

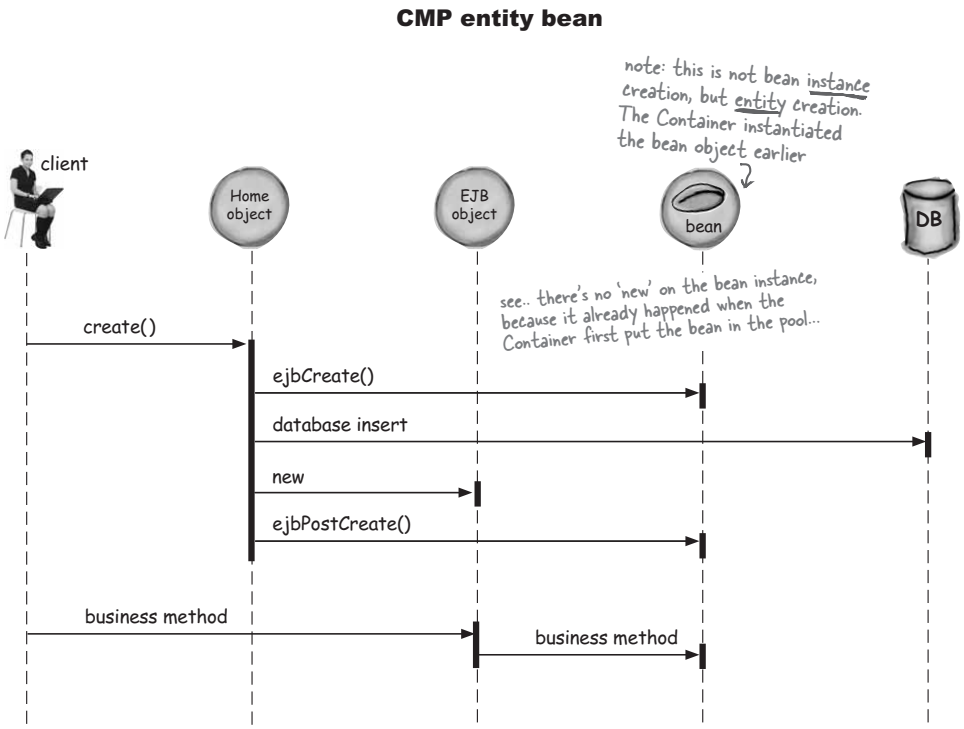
6 Container gives the EJB object and EntityContext the primary key value (#55).

7 Container calls ejbPostCreate("Billy Bob") on the bean, to give the bean a chance to finish initializing itself

8 The home returns the EJB object stub to the client

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

Creating a new entity



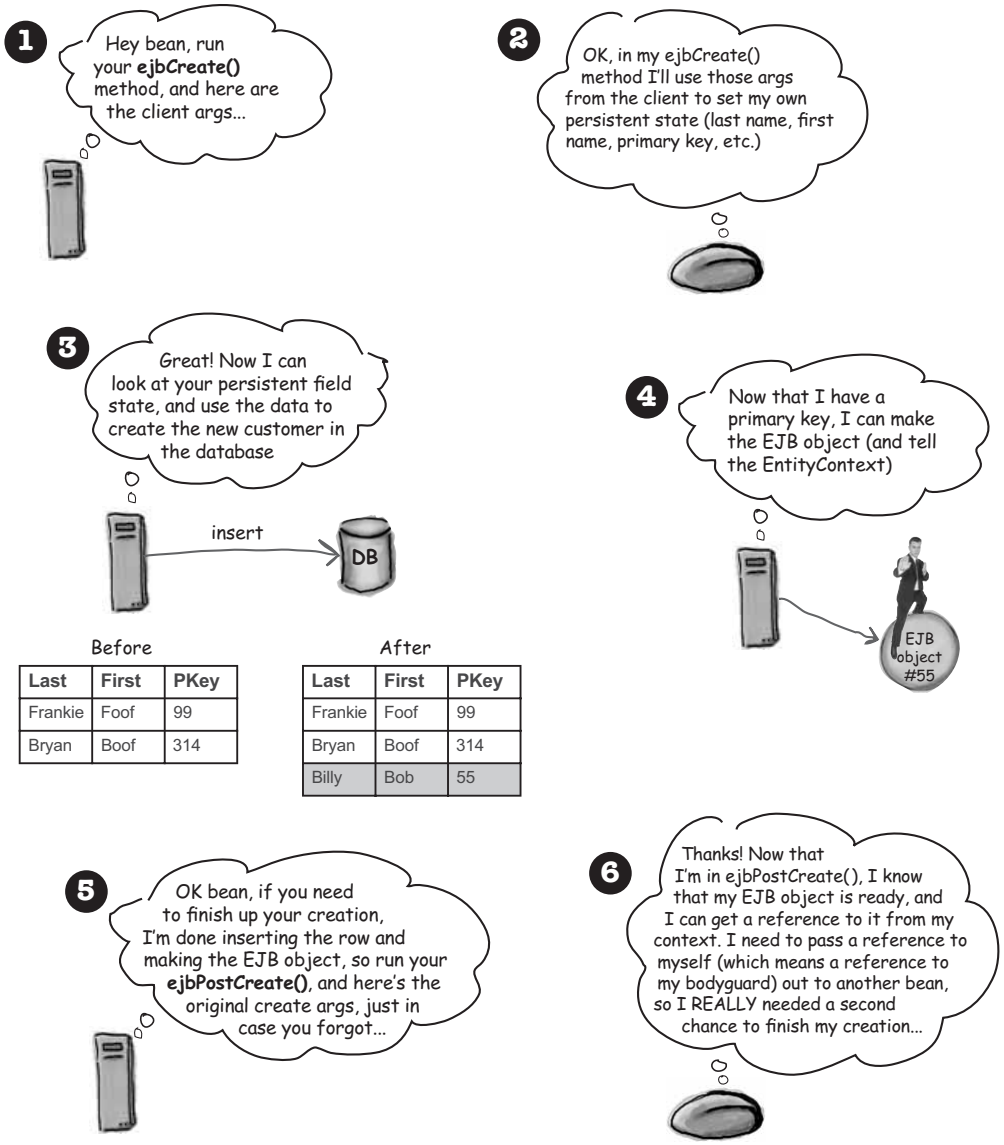
Whoa! This is quite different from session bean creation...

The Container calls ejbCreate() on a session bean instance only once, at the beginning of the bean's life. That's *one* big difference between session and entity beans—entity beans can run ejbCreate() over and over, like a business method. In fact, that's the best way to think of it... as just another home business method, as though it were named ejbInsert() (which, in our humble opinion, it should have been named, but once again, they forgot to ask *us*).

But the *second* difference between entity and session bean ejbCreate() methods is the order in which the EJB object is created. When an entity bean runs ejbCreate() there's no EJB object!! Yikes! That means the bean has no way to get a reference to its bodyguard. In other words, an entity bean cannot use the ejbCreate() method to, say, get a reference to its own EJB object and pass that reference to someone else. Remember, a bean can't pass a reference to itself, ever! When a bean wants to pass a reference to itself, it must pass a reference to its own EJB object. But it can't do that in ejbCreate(), because it's too early!

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

CMP Entity bean creation



Before

Last	First	PKey
Frankie	Foof	99
Bryan	Boof	314

After

Last	First	PKey
Frankie	Foof	99
Bryan	Boof	314
Billy	Bob	55

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

Implement your ejbCreate() methods

What to put in ejbCreate()

Put your entity initialization code here! This means setting values for your persistent fields... the fields that map to columns in the database row that's about to be inserted.

Notice we aren't saying, "put your *object* initialization code here..." With session beans, we told you to treat the ejbCreate() kind of like a constructor for the bean. But with entity beans, it's different because *the instance was initialized a long time ago* (when its setEntityContext() method was called).

So, ejbCreate() is about initializing the bean as this *new entity* the bean will now represent. For a CMP bean, this means the client has handed you arguments representing the data to insert into the database. The Container will look at the state of your persistent fields, once ejbCreate() returns, and use those fields to make the new row in the database.

If you do nothing else in ejbCreate(), ***you must make a primary key for this entity!*** That's your most important job. You might make the key based on arguments to the create method. You might make the key based on a key-generating algorithm. You might have a key server that you go to for the next key. Whatever it is, you must come up with the key, and assign it to your primary key field. When ejbCreate(), the Container looks at the state of your persistent fields, and uses that data to make the new row.

Warning! Remember, the Container calls your ejbCreate() before it can make the EJB object, so you can't use ejbCreate() to get a reference to yourself! (i.e. a reference to your bodyguard/EJB object). You'll have to wait until ejbPostCreate() before you can ask your context for a reference to your own component interface. You also can't use ejbCreate() to access your own persistent relationships; you have to wait until ejbPostCreate() (but we'll cover all that in the next chapter).

This must be the type of your primary key! In this case, the primary key we're using is a String, so that must be the return type (and we'll declare this in the DD)

```
public String ejbCreate(String last, String first) {
   // assign args to persistent fields
   this.setLastName(last);
   this.setFirstName(first);

   // set a primary key
   this.setPrimaryKey(this.makeKey());
   return null;
}
```

Once ejbCreate() completes, Container will use the values you set to make the new row.

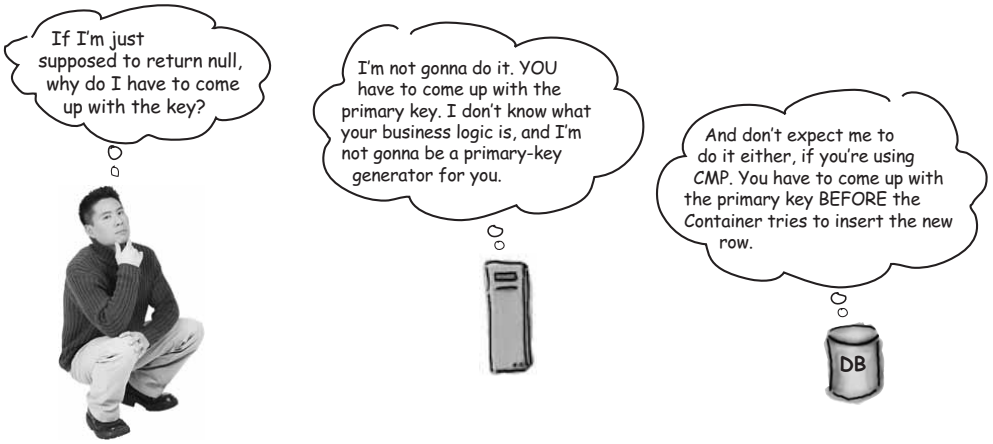
return null?? What's up with THAT?

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

entity primary key

Object identity: the primary key

Every entity MUST have a unique identity. The Container will never let you get away with having two or more entities with the same primary key. And as of EJB 2.0, coming up with that primary key is still ***your*** job. The Container won't do it (at least not according to the specification).



With CMP, you're still responsible for the primary key. That doesn't mean you won't use a primary key service of some kind. Maybe you have a service that automatically allocates a big block of primary keys from the database, and then hands them out as needed. Or you might have some type of unique identifier algorithm that makes unique keys for you. Or maybe you're using the customer's social security number or account number or... that's up to you to decide.

And the way you tell the Container what value the about-to-be-created entity should have is through the value of one or more of your persistent fields.

If you have just one field as your primary key, and it maps directly to a column in the database, you're set. But if you need more than one value to uniquely identify an entity (like, maybe it takes a combination of name and date), you can use a *compound* key that uses two or more of your container-managed persistent fields.

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

Rules for ejbCreate()

- You must implement the ejbCreate<method> to match each create<method> in the home interface.
- The method name must begin with the prefix "ejbCreate".
- The method must be declared public, and must not be declared static or final.
- The declared return type must be the entity bean's primary key type, even though you will return *null* from the method .
- The method arguments must be the same as the arguments of the matching create<method>.
- You may declare a throws clause with CreateException, or any arbitrary application (checked) exception that you like, as long as it was also declared in the home interface, but you must NOT declare a RemoteException.

Rules for Primary Keys

- A primary key class must be Serializable and public.
- You can use a single persistent field from your bean class as your primary key, by identifying both the field name and the class type in the DD.
- If you need two or more persistent fields to uniquely identify your entity, make a custom compound primary key class.
- A compound key class must be made up of fields that are defined as persistent fields in the bean class. The fields in the bean class must have public accessor methods.

By the end of ejbCreate() you MUST have a valid primary key assigned to the CMP field that you've the told the Container is your primary key field.

If you use a compound key, then ALL the fields that make up that key must have valid values.

If at the end of ejbCreate() your primary key is null, the Container won't do the create!

Q: How come I can't have the database come up with the primary key? That's how we usually do it. Are you telling me the Container can't do that?

A: Yes, that's what we're telling you. The contract you have with the Container says that by the time ejbCreate() is done, you've done whatever you had to do to make a valid primary key. You might have a server that lets you say, "Let the database come up with the key when you do the insert", but there's no guarantee in the spec, and you can't assume that all EJB vendors support this. Maybe some day in the future, there will be a deployment-independent way to say, "Use auto-generated key" in the deployment descriptor. But today is not that day. And EJB 2.0 is not that spec.

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

Implement your ejbPostCreate() methods

What to put in ejbPostCreate()

Finish your initialization code here! Now you can see why entity beans need an ejbPostCreate()—because ejbCreate() is too early for some things. The two most important reasons for ejbPostCreate() are that you can use it to get a reference to your own EJB object, and you must use it for accessing your container-managed relationships (CMR). (We'll look at CMR in the next chapter.)

Think of ejbPostCreate() as the other half of your ejbCreate(), with some Really Important Things happening in the middle. It's your second (and last) chance to finish your initialization, if you need to do things as part of creating the new entity that depend on having access to your EJB object (or something else that can happen only in ejbPostCreate(), and not in ejbCreate().

We'll revisit ejbPostCreate() in the next chapter, when we look at CMR.

Rules for ejbPostCreate()

- You must implement the ejbPostCreate<method> to match each create<method> in the home interface.
- The method name must begin with the prefix "ejbPostCreate".
- The method must be declared public, and must not be declared static or final.
- The declared return type must be void!
- The method arguments must be the same as the matching ejbCreate<method>.
- You may declare a throws clause with CreateException, or any arbitrary application (checked) exception that you like, as long as it was also declared in the home interface, but you must NOT declare a RemoteException.

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

All three have the same args, but different return types!

Home interface
create<method>

```
public Customer create(String last, String first)throws
                                    CreateException,RemoteException);
```

notice the name differences

Bean class
ejbCreate<method>

```
public String ejbCreate(String last, String first) {...}
```

another difference

Bean class
ejbPostCreate<method>

```
public void ejbPostCreate(String last, String first){...}
```

the bean class doesn't have to declare the same exceptions, unless it throws them. (And you NEVER throw a RemoteException from the bean. Only the Container throws a RemoteException.

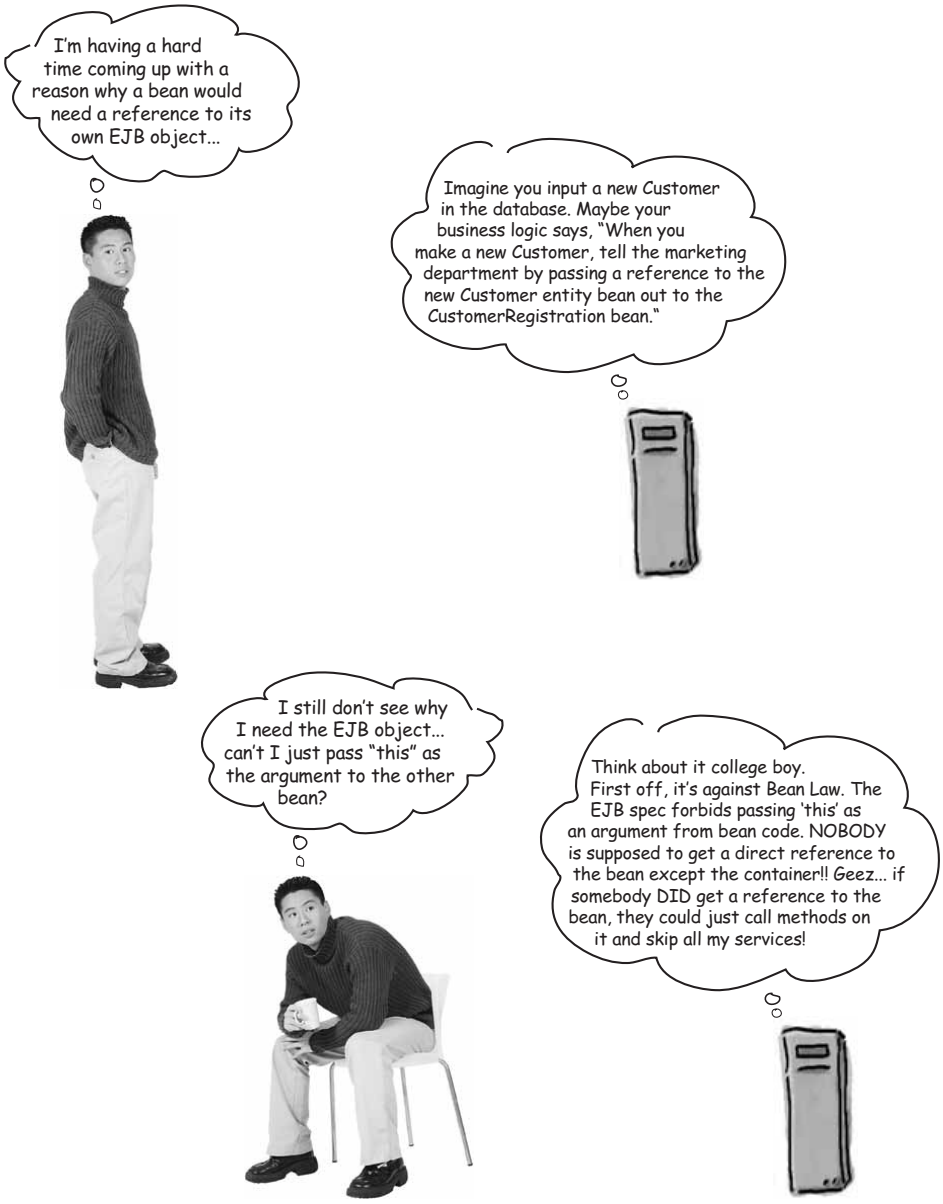
Think about the three methods above, and how they're related to one another. Think about what they're each responsible for. Now answer these two questions:

1. Why are the return types declared like this? In other words, explain why each of the three different return types is what it is...

2. Why do both ejbCreate<method> and ejbPostCreate<method> have the same arguments?

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

***entity bean** creation*



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

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

***entity bean** creation*

Bean things you can do during entity creation:

timeline

ejbCreate()

Use your EntityContext to:

- [x] get a reference to your home
- [ ] get a reference to your EJB object
- [ ] get your primary key
- [x] get security information about the client
- [x] force a transaction to rollback (CMT beans)
- [x] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [x] another bean's methods
- [x] a resource manager (like a database)

it's too early to get a reference to your EJB object or primary key, because the container is still waiting for you to finish your ejbCreate(). Once you return from ejbCreate(), the container will look in your bean and see what state changes you've made as a result of the initialization you did based on the client-supplied args. Only THEN can the container figure out what your primary key is (by looking in your primary key field!), and it needs that to make an EJB object.

ejbPostCreate()

Use your EntityContext to:

- [x] get a reference to your home
- [x] get a reference to your EJB object
- [x] get your primary key
- [x] get security information about the client
- [x] force a transaction to rollback (CMT beans)
- [x] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

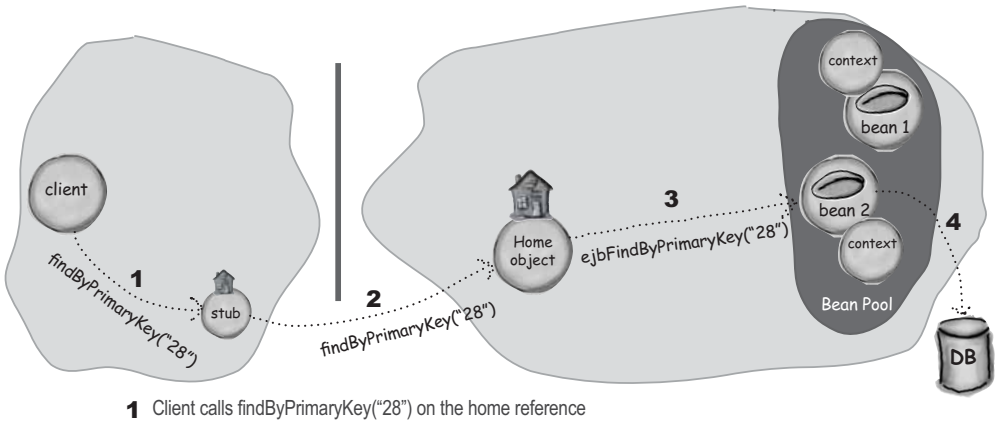
- [x] your special JNDI environment
- [x] another bean's methods
- [x] a resource manager (like a database)

By the time you're in ejbPostCreate(), the container knows your primary key and has made (or found) the EJB object for this key.

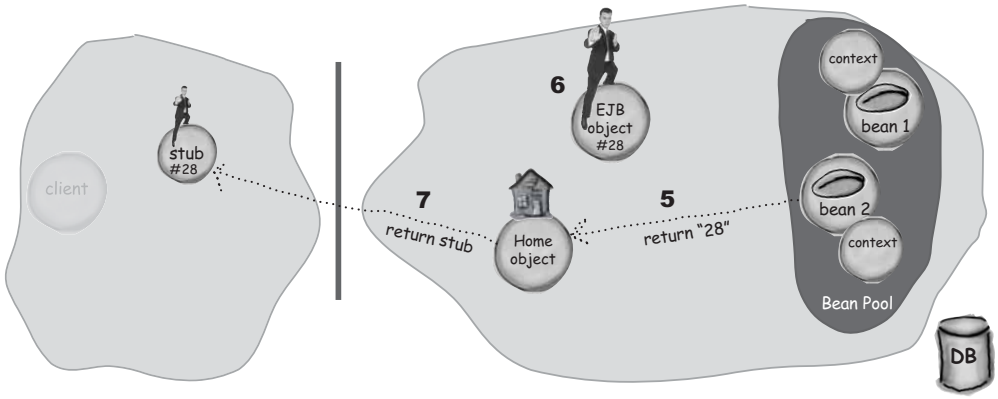
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

CMP Entity finder

Scenario: client wants to get a reference to an *existing* entity bean



1. Client calls findByPrimaryKey("28") on the home reference
2. The finder method is passed to the home object
3. A bean is selected from the pool to run the ejbFindByPrimaryKey("28") method
4. The bean does a select on the database, to verify that an entity with primary key #28 exists



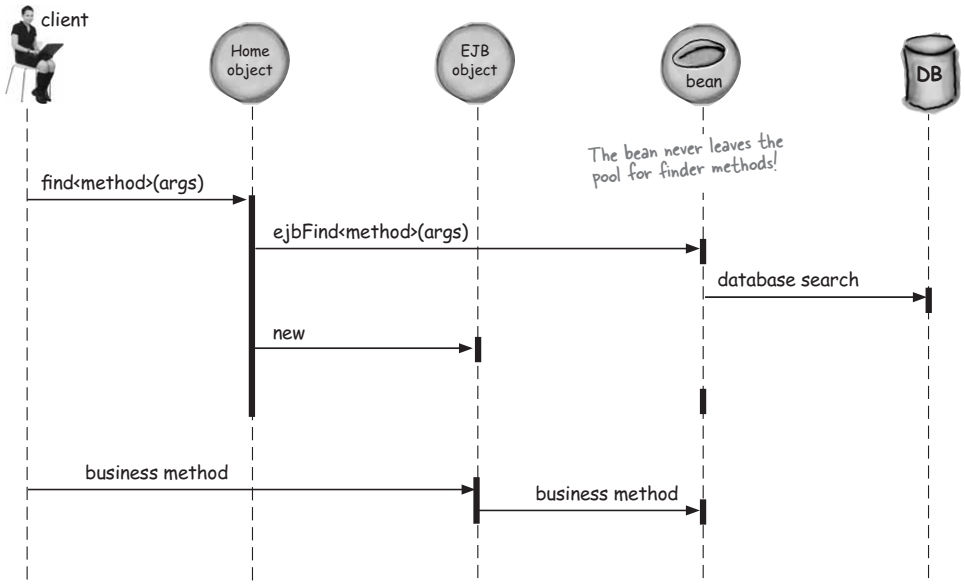
5. The bean returns the primary key (#28) to the home, which means the entity exists in the database
6. The Container makes (or finds) an EJB object for #28
7. The Container returns the EJB object stub for #28

Note: if the bean had not found a matching entity for primary key #28, the client would get an ObjectNotFoundException (subclass of FinderException)

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

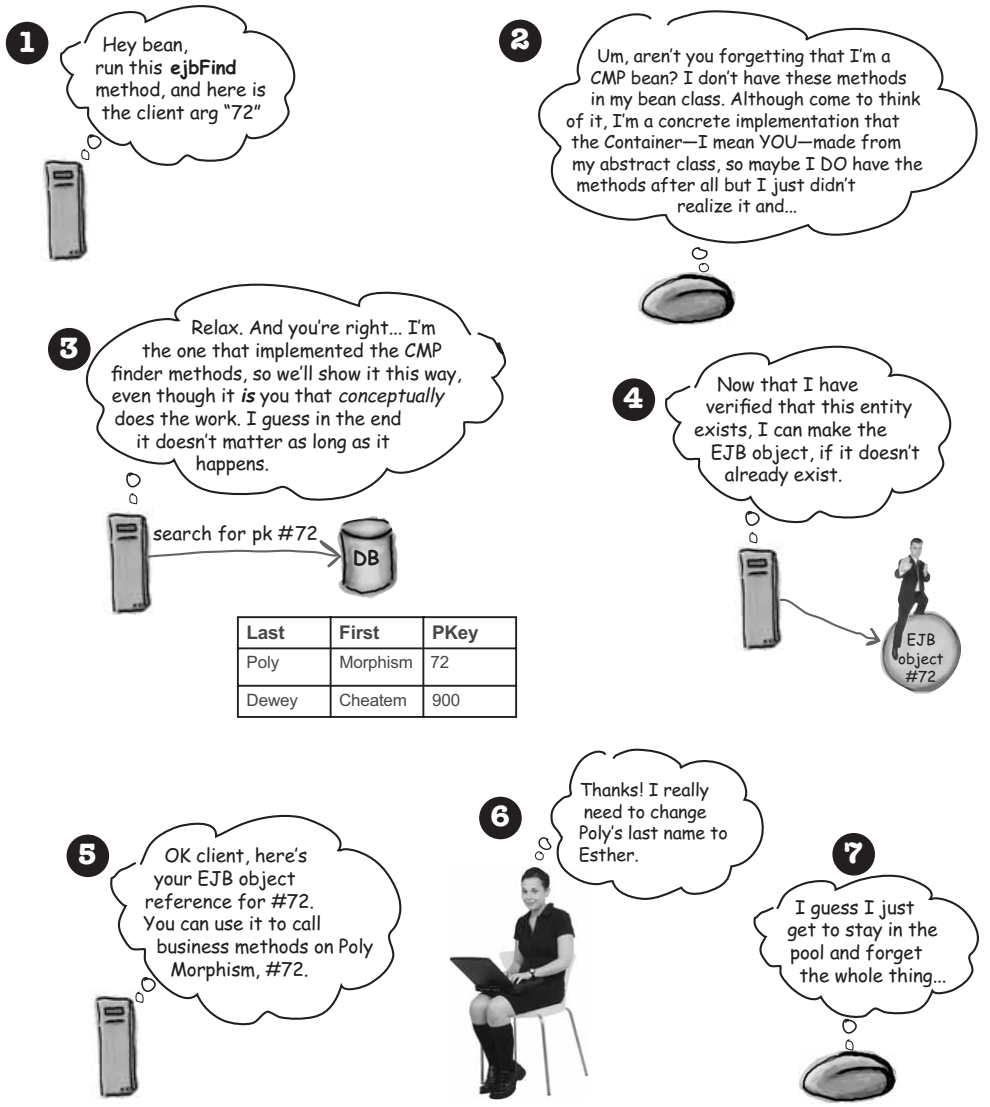
Finding an existing entity

CMP entity bean



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

CMP Entity bean finders

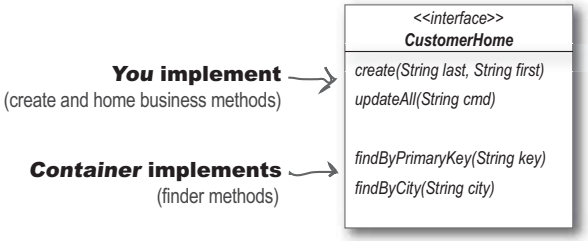


Last	First	PKey
Poly	Morphism	72
Dewey	Cheatem	900

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

entity bean finders

YOU don't implement the Finder methods!



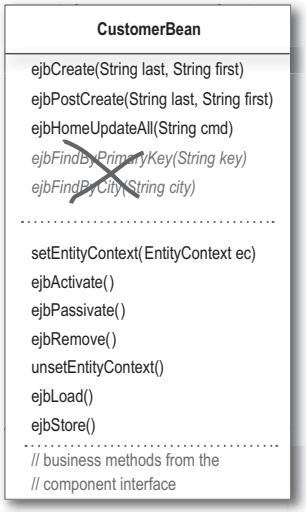
Don't put anything in your bean class about the finders. The Container implements your finder methods, using the home interface and your deployment descriptor to figure out what to do.

Your bean class must not even MENTION the finder methods.

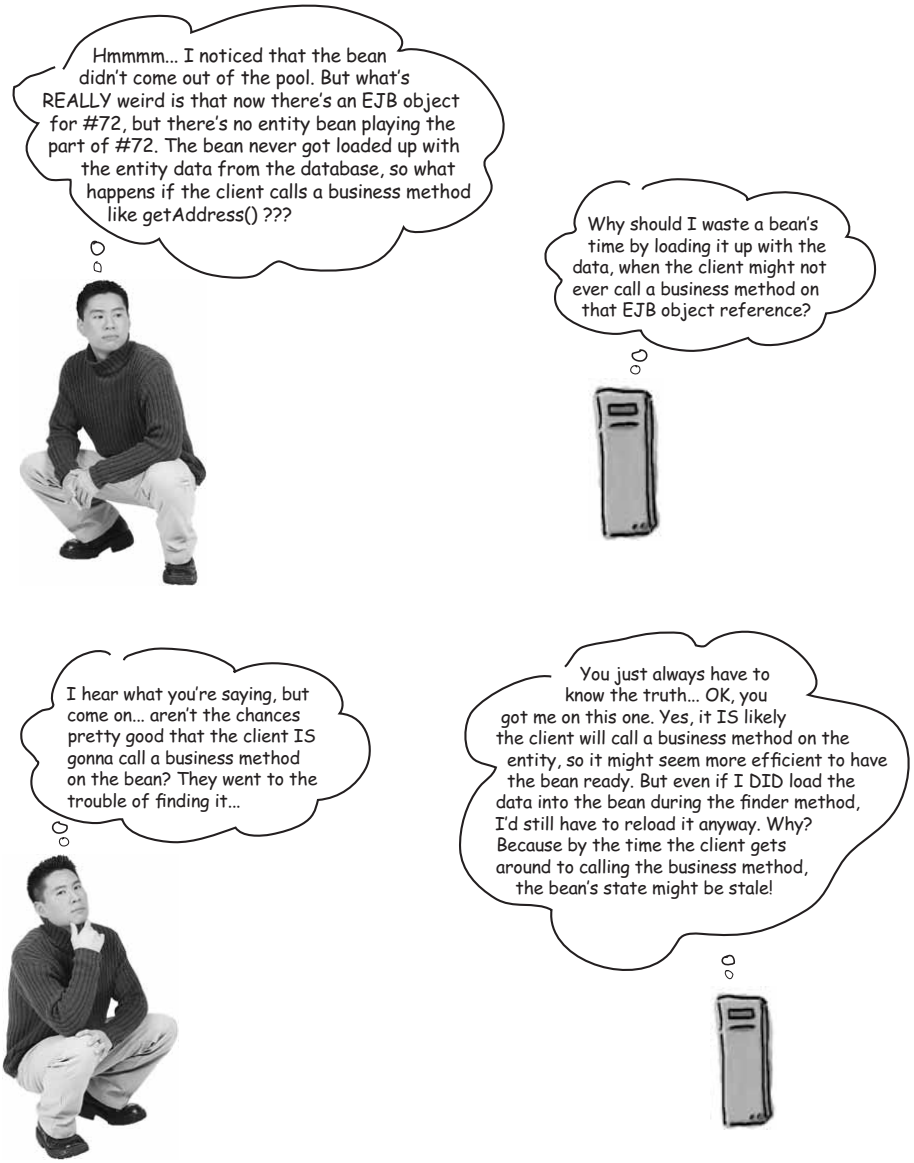
Just don't write 'em. For a CMP bean, don't put anything in your class about the finder methods. You will implement your create and home business methods, but as far as your own bean class—the one that you write—goes, there are no finder methods.

You do declare them in the home interface, of course, including the mandatory findByPrimaryKey(), but you don't write any other Java code related to your finders.

The Container looks in your home interface and your deployment descriptor to figure out what to do with the finder methods, and the Container writes all the code for them. You'll never see it.



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



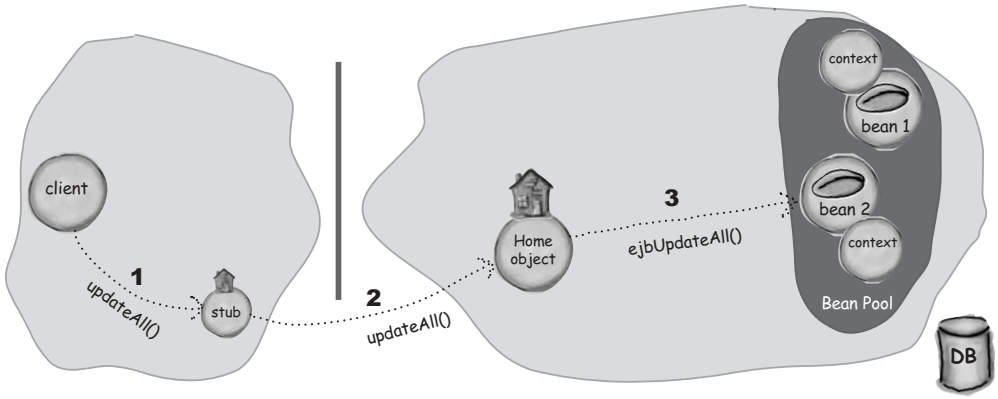
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

***entity bean** finders*

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

CMP Home Business Methods

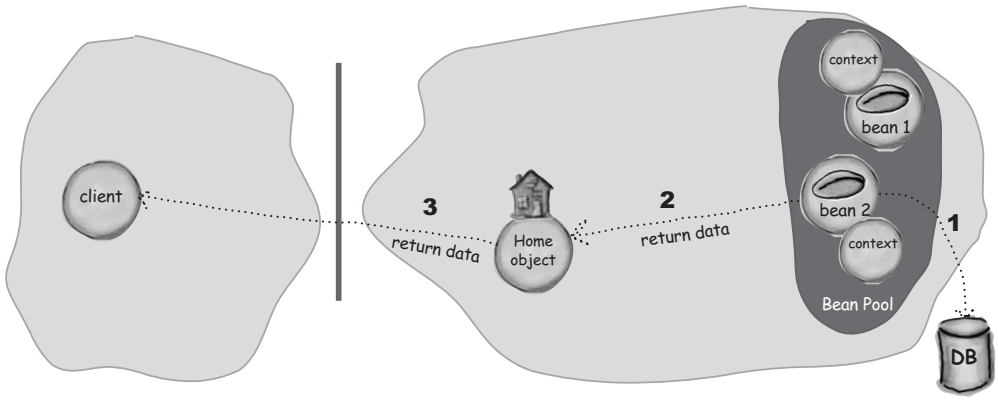
Scenario: client wants to call a business method in the home



1 Client calls updateAll() on home reference.

2 The updateAll() method is passed to the home object.

3 A bean is selected from the pool to run the ejbUpdateAll() method.



1 Without leaving the pool, the bean runs the home method, probably calling on a Container-implemented select<method> to access the database.

2 The bean returns from the method (possibly returning data).

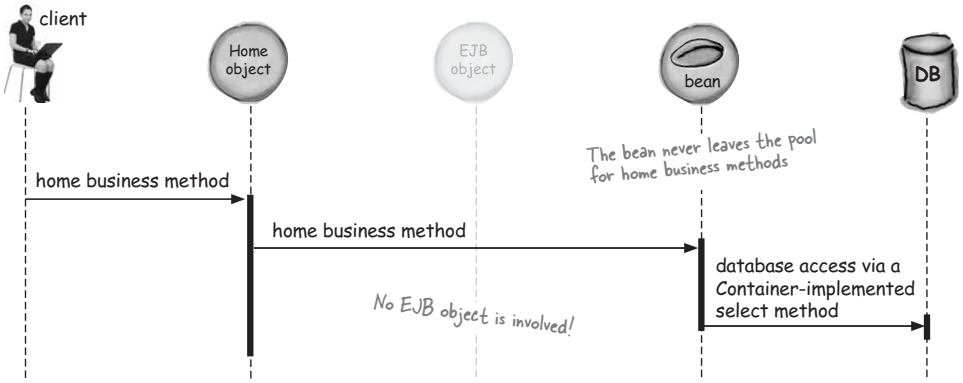
3 The Container passes the return value back to the client.

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

home business methods

Client invokes a Home Business Method

CMP entity bean



What to put in a home business method

As you write your code, remember that an entity bean stays in the pool when it runs a home business method. Put code in your home business methods that apply to the underlying database—for a group of entities rather than one specific entity—and that don't return the bean's component interface (the way finders and create methods must).

```
public Collection ejbHomeDisplayAll() {
   // do a select on the database, using a container-implemented
   // select method (next chapter), then return an ArrayList of Strings
}
```

Rules for home business methods

- You must have an ejbHome<method> method for every home <method> in the home interface.
- The name must begin with the prefix "ejbHome", followed by the name of the home <method>, but with the first letter of the method capitalized. For example, updateAll() in the home interface will be ejbHomeUpdateAll() in the bean class.
- The method must be declared public and NOT static.
- If the home interface is Remote, the arguments and return values must be legal types for RMI-IIOP.
- You may declare a throws clause with your own application (checked) exceptions, as long as the exceptions were also declared in the home interface.
- Even if your life depends on it, you must NOT declare a RemoteException.

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

Bean things you can do in home business methods

Use your EntityContext to:

- [x] get a reference to your home
- [ ] get a reference to your EJB object
- [ ] get your primary key
- [x] get security information about the client
- [x] force a transaction to rollback (CMT beans)
- [x] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [x] another bean's methods
- [x] a resource manager (like a database)

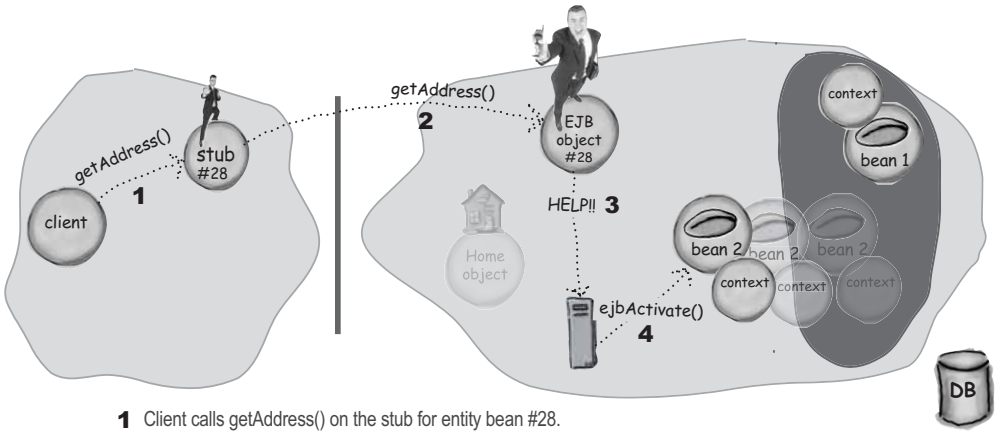
two things you can't do because the bean isn't BEING an entity! The bean is in the pool acting on behalf of ALL the entities of this type, but not any one particular entity. So the bean has no EJB object and of course, no primary key.

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

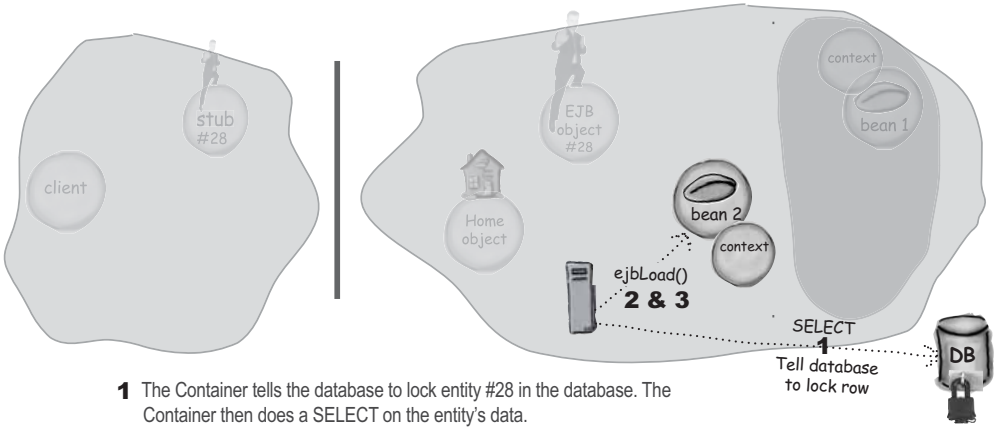
entity business methods

Starting a business method (in a transaction)

Scenario: client wants to get the address of a specific CMP Customer entity



1. Client calls getAddress() on the stub for entity bean #28.
2. The call is passed to the EJB object.
3. The EJB object gets the call and panics, because there IS no entity bean for #28!
4. The Container sees that this method needs a transaction, so the Container starts one, then pulls a bean from the pool, and calls ejbActivate() on the bean

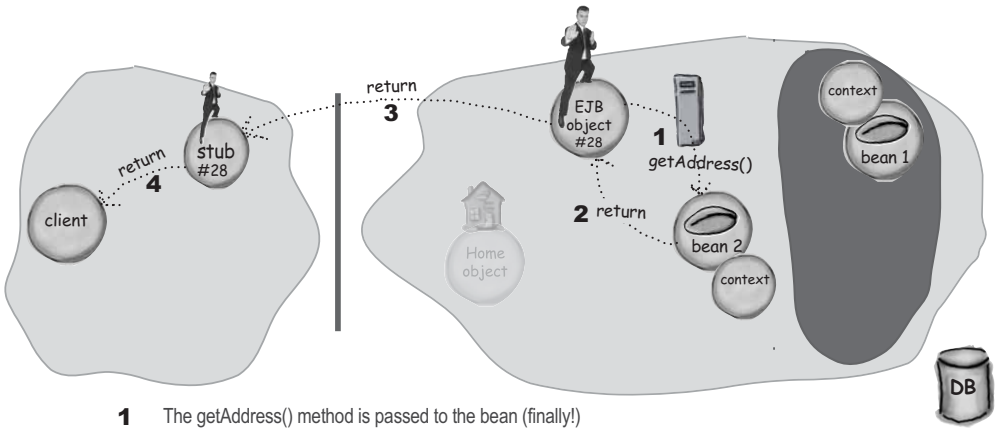


1. The Container tells the database to lock entity #28 in the database. The Container then does a SELECT on the entity's data.
2. The Container populates the entity bean's persistent fields with the real entity data.
3. The Container calls ejbLoad() on the bean, to tell the bean, "You've just been loaded."

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

Completing a business method (and a transaction)

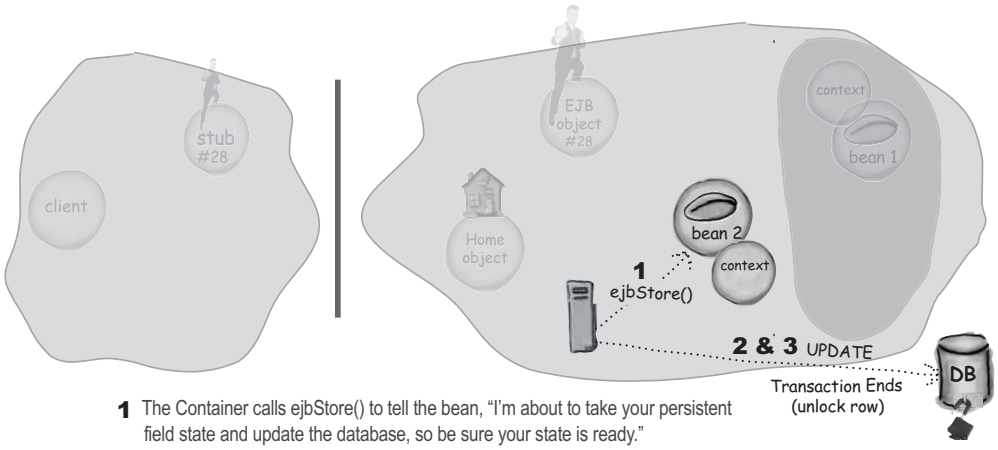
Scenario: bean completes a business method that ends a transaction



1 The getAddress() method is passed to the bean (finally!)

2 The bean returns from the method

3/4 The EJB object sends the return back to the client



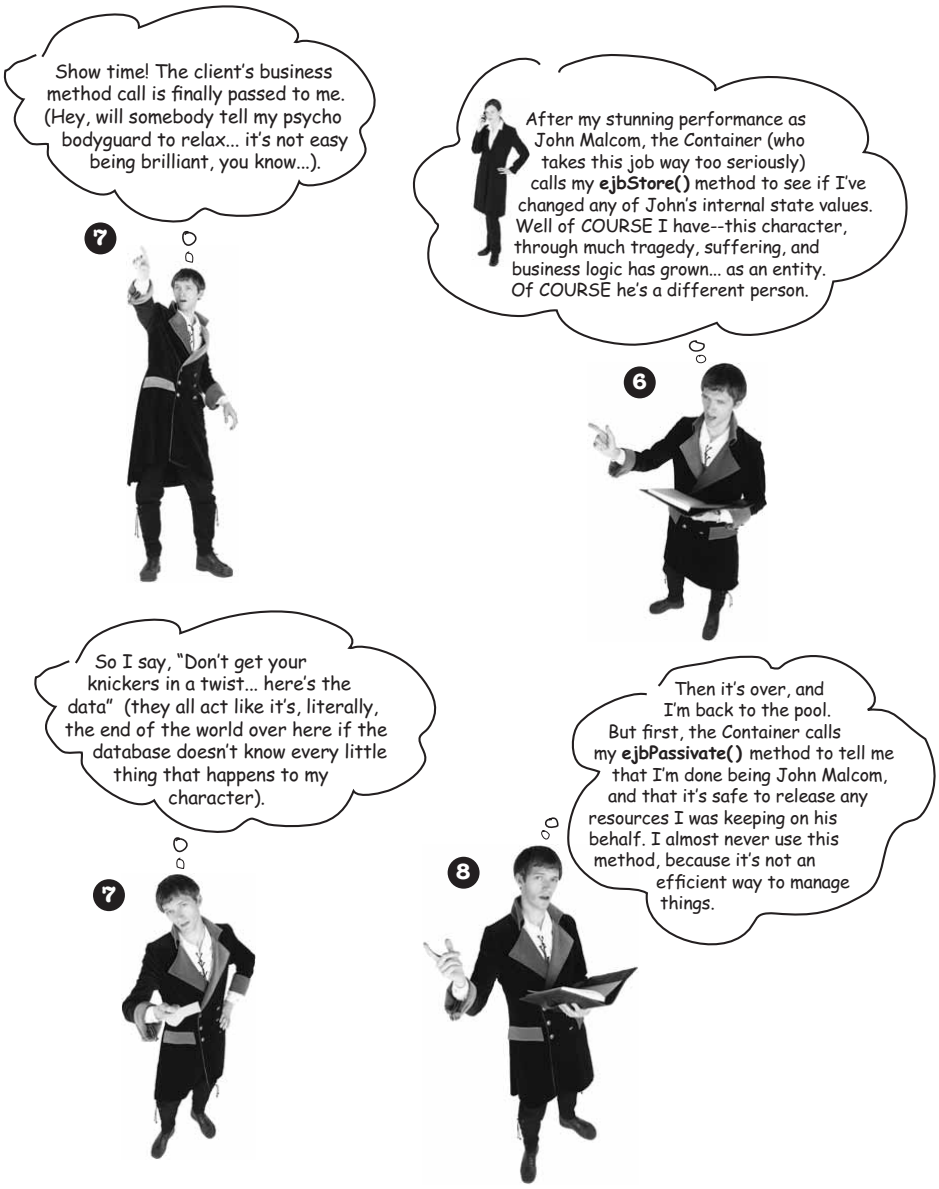
1 The Container calls ejbStore() to tell the bean, "I'm about to take your persistent field state and update the database, so be sure your state is ready."

2 The real entity is updated in the database.

3 The Container commits the transaction and tells the database to unlock the entity.

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

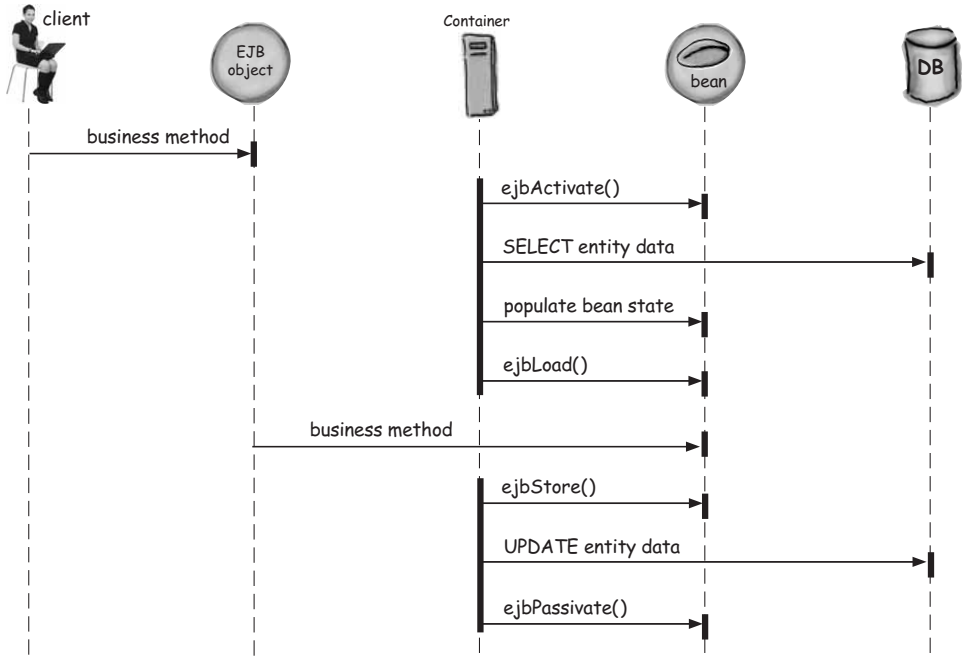
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01



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

activation *passivation*

Activation/passivation of a CMP entity bean



The Container always calls ejbLoad() after ejbActivate().

The Container always calls ejbStore() before ejbPassivate().

The Container can call ejbStore() and ejbStore() at other times.

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

If you're taking the exam, trust us here, you REALLY need to stop right now and do this exercise, before turning the page. Think about everything you've learned in this chapter about the lifecycle of an entity bean, especially how and when it comes out of the pool to *become* an entity.

Take your best guess about which aspects of beanness are available during each of the four container callbacks for ejbActivate(), ejbPassivate().

Use your EntityContext to:

ejbActivate()	ejbPassivate()	ejbLoad()	ejbStore()	
❑	❑	❑	❑	get a reference to your home
❑	❑	❑	❑	get security information about the client
❑	❑	❑	❑	force the transaction to rollback
❑	❑	❑	❑	find out if the transaction has already been set to rollback
❑	❑	❑	❑	get a reference to your EJB object
❑	❑	❑	❑	get your primary key

Access:

ejbActivate()	ejbPassivate()	ejbLoad()	ejbStore()	
❑	❑	❑	❑	methods of another bean
❑	❑	❑	❑	your special JNDI context
❑	❑	❑	❑	a resource manager (like a database)

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

activation *passivation*

Bean things you can do during activation and loading

timeline

ejbActivate()

Use your EntityContext to:

- [x] get a reference to your home
- [x] get a reference to your EJB object
- [x] get your primary key
- [ ] get security information about the client
- [ ] force a transaction to rollback (CMT beans)
- [ ] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [ ] another bean's methods
- [ ] a resource manager (like a database)

The Container prefers (no, make that INSISTS) that if you access a bean or a resource manager, you must be in a 'meaningful transaction context', so ejbActivate() is too early. But once you get to ejbLoad(), you're IN a transaction (unless you've told the Container NOT to use a transaction... which is for very special cases only).

ejbLoad()

Use your EntityContext to:

- [x] get a reference to your home
- [x] get a reference to your EJB object
- [x] get your primary key
- [x] get security information about the client
- [x] force a transaction to rollback (CMT beans)
- [x] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [x] another bean's methods
- [x] a resource manager (like a database)

Now we can do everything that an entity bean CAN do... the same things we can do in a business method. You can think of this as the beginning of the business method.

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

Bean things you can do during passivation and storing

timeline

ejbPassivate()

Use your EntityContext to:

- [x] get a reference to your home
- [x] get a reference to your EJB object
- [x] get your primary key
- [ ] get security information about the client
- [ ] force a transaction to rollback (CMT beans)
- [ ] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [ ] another bean's methods
- [ ] a resource manager (like a database)

By the time the Container starts to passivate you, you're no longer associated with a client or a transaction

Just as it was with ejbActivate(), the Container no longer has you in a 'meaningful transaction context' (in other words, you can't be in a transaction at this point), so you can't get a reference to a bean or a resource manager. The Container says it's too dangerous to do this without a 'meaningful transaction context'. (see the transactions chapter)

ejbStore()

Use your EntityContext to:

- [x] get a reference to your home
- [x] get a reference to your EJB object
- [x] get your primary key
- [x] get security information about the client
- [x] force a transaction to rollback (CMT beans)
- [x] find out if the transaction has already been set to rollback (CMT beans)
- [ ] get a transaction reference, and call methods on it (BMT beans only, so entities can't use this)

Access:

- [x] your special JNDI environment
- [x] another bean's methods
- [x] a resource manager (like a database)

FYI -- this page is EXACTLY the same as the previous one on activation and loading. (except for the notes we made). The rules for what's allowed during activation are the same for passivation, and the rules for loading are identical to the rules for storing.

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

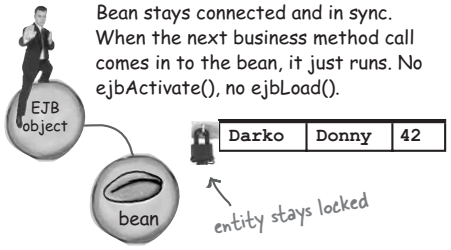
Commit options: what REALLY happens to a bean after a transaction commits?

(the previous pictures are just ONE way it could work, but there are two other ways)

1 Commit option A

The bean stays ready, attached to the EJB object, loaded with data but NOT in a transaction. The Container keeps the entity locked, so that nobody can change the bean's state. When a client calls a business method on this entity bean's EJB object, the Container does NOT do an ejbActivate() or ejbLoad(), and assumes the bean's data is still in sync with the underlying persistent store.

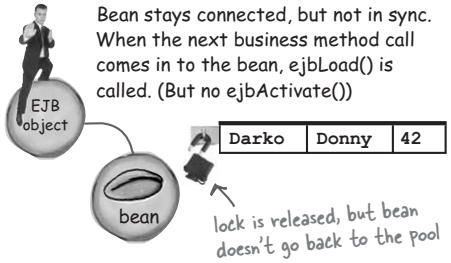
A



2 Commit option B

The bean stays attached to the EJB object, loaded with data but the bean's state is marked as 'invalid'. In other words, the Container knows that the bean might become stale between now and the next time a client invokes a method on this bean, so the Container will do an ejbLoad() when a new business method comes in for this entity. (But no ejbActivate()).

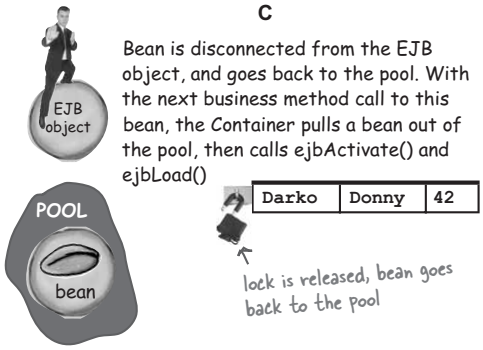
B



3 Commit option C

The bean is passivated and put back into the pool. The next time a client calls a business method on this entity, everything we've seen in this chapter happens—a bean comes out of the pool, is activated, the loaded, and finally the business method it passed to the bean.

C



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

there are no Dumb Questions

Q: How does these commit options affect me? Do I need to know which one my Container is choosing? Do I get a choice?

A: According to the specification, your vendor is free to use any of the three options. Your container might let you choose (or tune) the option, but don't always count on it. But do you need to know which is being used?

You can see where the trade-offs are, of course—if you keep the entity locked in the database, no other applications can use that entity's data. That's fine if the EJB is the only application that needs that database. And that option (commit option A) gives you the best performance when a business method call comes in for the entity, because there's no ejbActivate() (the bean never went back to the pool), and more significantly—there's no trip to the database or an ejbLoad()!

Q: But couldn't option A really kill me, if I want to keep the entities unlocked in the database when they're not being used by the bean?

A: For this reason, most vendors won't use option A as a default; they allow it only if the deployer or admin chooses it (assuming it's even supported at all).

Q: I see another, possibly even WORSE problem with option A... doesn't this mean a continually growing heap full of objects? Like, each time someone uses a particular entity, an entity bean instance is created that then stays stuck to the EJB object? Aren't pools more efficient because you need only enough beans to service the actual in-progress methods?

A: No container worth configuring would leave you stuck with a constantly, endlessly growing accumulation of objects. Even a container that does use option A can use something like a LRU (Least Recently Used) algorithm to say to itself, "Hmmm... nobody has called a method on entity #908 for a long time, so I'll passivate that bean and put him back in the pool (and unlock the row)." But, with at least the option of using option A, a container can say to itself, "This entity #405 is really popular. It's a big pain to keep going to the database and doing the whole load/store/activate/passivate cycle each time, so I'll just keep him loaded and ready, even between transactions."

Q: Oh man, I just thought of something REALLY bad with this architecture...even if you do NOT keep accumulating entity beans, what about the EJB objects??? Don't they just keep growing and growing so that each time a client does a find, and gets back a stub, an EJB object is created for that entity, and it just stays around until the entity itself is removed? At any given time, I will have all the EJB objects on the heap that represent every entity anyone has ever accessed up to that point...

A: Ah... now we come to the difference between the "conceptual" view of the architecture, and the "actual" implementation.

According to the spec, you're supposed to assume that this is indeed how it works: when a client wants an entity bean, either through a create or find, the Container makes an EJB object for that entity ("here's the EJB object for #24, here's the EJB object for #98", etc.). And as a Bean Provider (as opposed to, say, a container/server provider), you are to program as if that's really the way it works.

But internally, the Container might be doing something quite different from the conceptual architectural view. Imagine that YOU were a container developer—how would you implement things?

One idea might be to put the burden on the clients, by keeping all the information in the stubs. For example, when the client gets a stub to an EJB object, you can make sure that the stub knows the unique ID for the entity (i.e. the primary key). But the EJB object's might be generic, in such a way that when the client calls a method on a stub, that stub is connected not to an EJB object just for that entity, but rather a one-EJB object-fits-all that knows how to make sure the method ends up at the right entity. That way, when a client isn't using an EJB object, that EJB object isn't sitting around wasting space. That's just one idea.

Q: That might be fine for stubs, but what if the client is local? Then they have a real reference to the EJB object, so that EJB object would HAVE to be entity-specific.

A: Yes, that's true. But with local clients, normal Java garbage collection works. When there are no clients with references to a particular EJB object, the instance will die.

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

there are no Dumb Questions

Q: Which of the three commit options do you think is most used?

A: Well... we can't really answer that, but if we were in a betting mood, our money would be on option B. A 'smart' option B, that knows when it makes sense to hang on to a bean and when it doesn't. But avoiding the ejbActivate(), which is usually useless anyway, is a Good Thing.

Q: But isn't the hit to the database—the load—a much bigger hit than just one more method call on the stack?

A: First of all, activation is probably a little more than just one more method call on the stack, because there's overhead just in maintaining the organization of the pool itself. But yes, you're right about the database hit being a bigger deal. But... there's all sorts of ways the Container can optimize, and what's the alternative? The only alternative (option A) keeps the database locked, which is typically *not* what you want.

Q: But what if my EJB app IS the only thing that uses that database?

A: I think we mentioned this earlier...if you know absolutely positively no question that your EJB app is the only thing touching that database, then yes, if your vendor supports it, you'll have better performance with option A. Of course, caching things in memory is still dangerous, because entities are persistent! So the vendor still must implement something that saves the state of the entities to the database, if they change. So there will still be storing, if not loading.

There are still other potential problems with option A, though. For example, you might have a server that supports clustering, and the server will have to be certain that all instances of that entity bean in sync, not just with the database, but with one another. But that's another issue. You'll learn more in the patterns and performance chapter.

Q: So is the real point of all this the fact that as long as you're in a transaction, you won't have all these load/store activate/passivate calls?

A: Yes, a big *part* of the point, anyway.

Q: Then shouldn't your goal be to have the longest transactions?

A: You aren't being serious, right? Just for fun, we'll pretend you are, and say, NO NO NO NO. Think about your concurrency. A good rule of thumb (with a zillion exceptions, of course so all rule-of-thumb disclaimers apply) is that transactions should be no longer than absolutely necessary. The longer your bean is in a transaction, the more likely it is that others are waiting in line to get to that entity! Of course,

it's also possible to have transactions which are *too short*, which means you're doing more loads/stores than you need to, so the key is to have your transactions be as long as you *need*, and *no longer*.

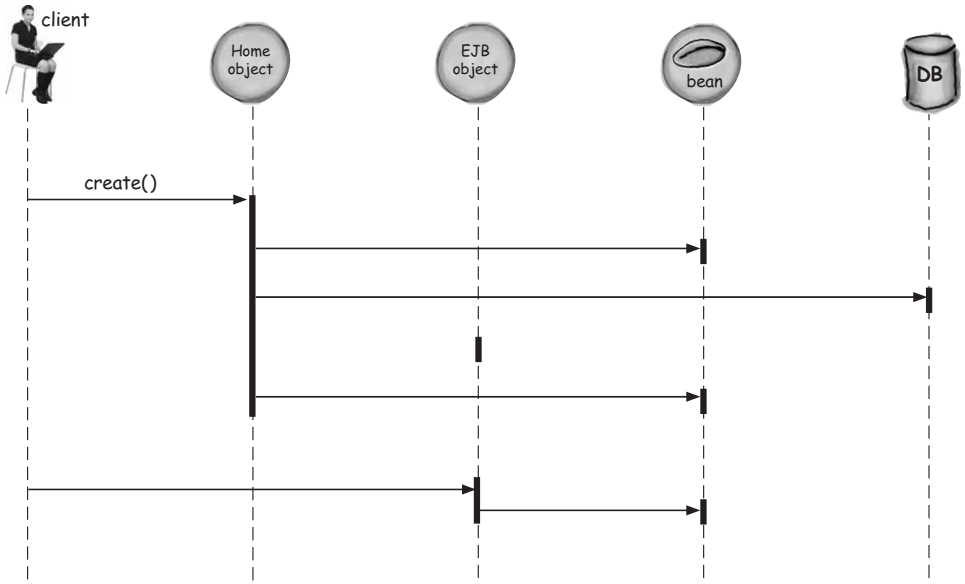
Q: One more thing... doesn't this mean that ejbPassivate and ejbActivate might not ever be called on a bean, if the container uses option A or B? And doesn't this mean that I shouldn't rely on ejbActivate or ejbPassivate in my design?

A: YES! That's EXACTLY what it means. It's usually a lousy strategy. But you can imagine how it might *sound* like a good idea...

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

CMP entity bean Creation

Fill in the five missing arrow labels for the object interaction diagram. Remember, they aren't necessarily method calls. This is exactly the same diagram that you saw earlier in the chapter. Except yours will look nicer, because you'll probably make it all fancy with exotic symbols and maybe color-coded markers and...

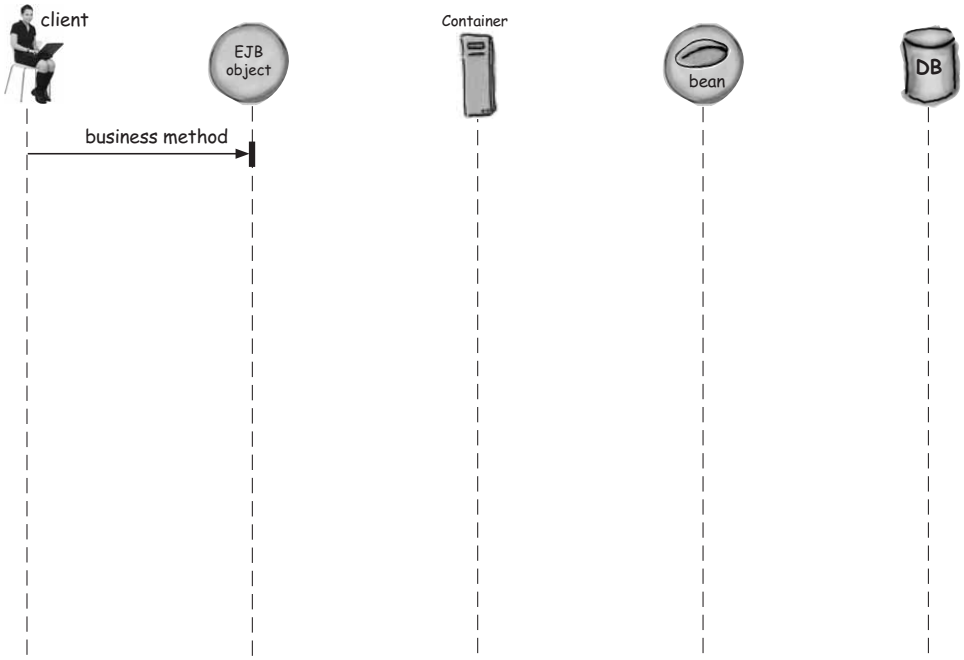


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

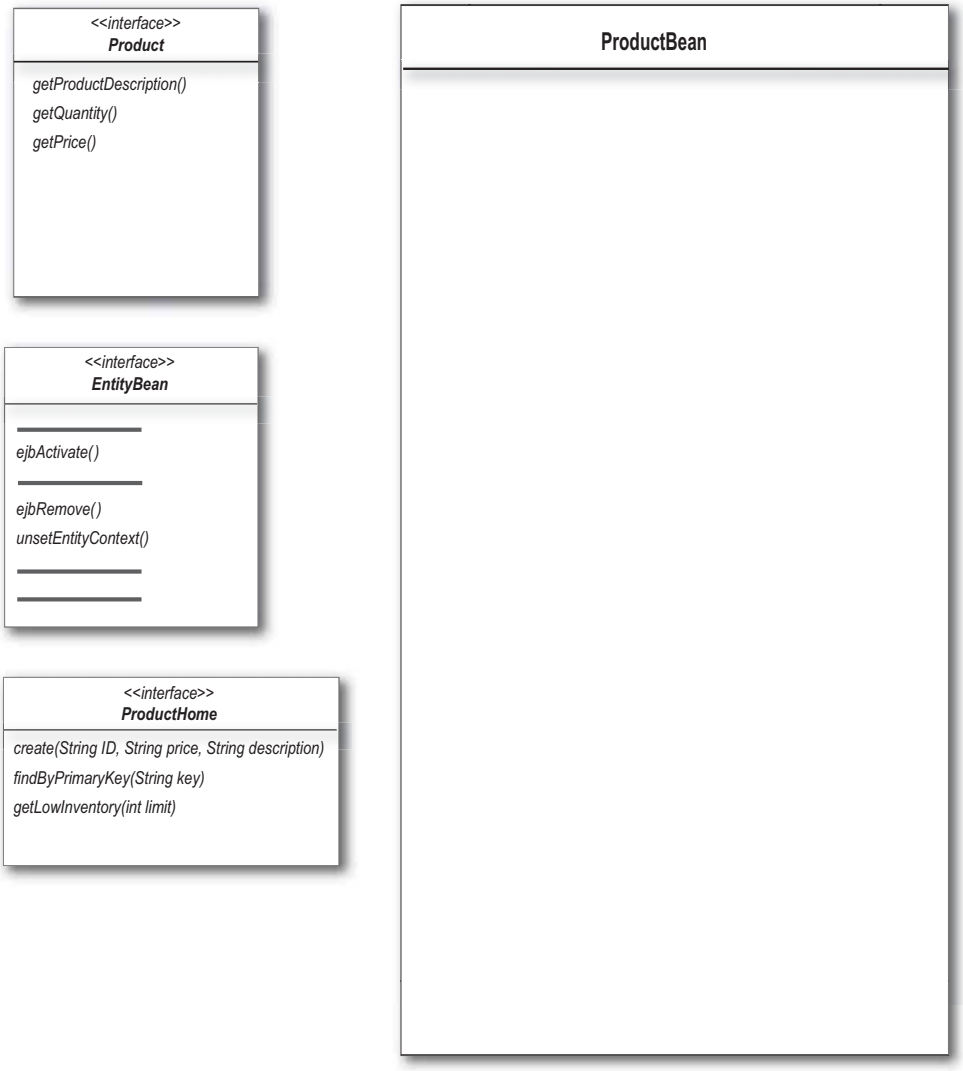
Entity Bean Business Method (with activation & passivation)

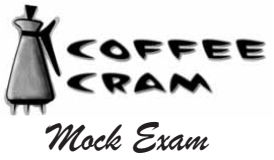
This time, you're gonna draw the arrows yourself! Assume that the bean is in the pool at the time the client calls a business method (which also means the bean is not in a transaction). Draw arrows and label them with the actions that occur through the completion of the business method. Show what happens when the bean returns to the pool.

When we did it, there were eight more arrows, but you might have more or less depending on how you want to do it. Do NOT flip back through the most recent pages. You can do this.



Fill in the ProductBean UML-ish box with the methods that YOU must write in your bean class, given the component and home interfaces. Don't forget the container callbacks from EntityBean, although we've shown you only three of the seven. The rest you'll have to remember and fill in.





1 What's true for a bean provider when creating an entity bean using container-managed persistence? (Choose all that apply.)

- ❑ A. Container-managed persistent fields must be defined in the entity bean class.
- ❑ B. Container-managed relationship fields must be defined in the entity bean class.
- ❑ C. When implementing a one-to-many relationship, the java.util.List interface must not be used.
- ❑ D. Accessor methods for container-managed relationship fields must be exposed in the bean's remote component interface.

2 Which of the following is a legal accessor method for a persistent field in an entity bean with container-managed persistence? (Choose all that apply.)

- ❑ A. `public getCustomerNum();`
- ❑ B. `public void getCustomerNum();`
- ❑ C. `abstract void getCustomerNum();`
- ❑ D. `public abstract int getCustomerNum();`

3 Which of the following are legal accessor method(s) in an entity bean with container-managed persistence? (Choose all that apply.)

- ❑ A. `public abstract int GetCustomerNum();`
- ❑ B. `public abstract int getcustomerNum();`
- ❑ C. `public abstract int getCustomerNum();`
- ❑ D. `public abstract int getCustomerNum() { };`

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

4 Which are requirements for a CMP entity bean class? (Choose all that apply.)

- ❑ A. The class must define a **`finalize()`** method.
- ❑ B. The source file must define at least one constructor.
- ❑ C. The class must be declared public and abstract.
- ❑ D. The class must implement, directly or indirectly, the **`javax.ejb.EnterpriseBean`** interface.
- ❑ E. All getter and setter methods for the bean's abstract persistence schema must be abstract.

5 Which are legal declarations for a CMP bean's ejbCreate methods? (Choose all that apply.)

- ❑ A. `public void ejbCreateBigCustomer() throws javax.ejb.CreateException`
- ❑ B. `public String ejbCreateAccount() throws javax.ejb.CreateException`
- ❑ C. `static String ejbCreate() throws javax.ejb.CreateException`
- ❑ D. `public int ejbCreate() throws javax.ejb.CreateException`
- ❑ E. `public final String ejbCreate() throws javax.ejb.CreateException`

6 Which are legal declarations for a method in a CMP bean? (Choose all that apply.)

- ❑ A. `public Account ejbSelectAcct(long x) throws javax.ejb.FinderException`
- ❑ B. `public abstract Acct ejbSelectAcct(long x) throws javax.ejb.FinderException`
- ❑ C. `public Account ejbPostCreate(Acct key) throws javax.ejb.CreateException`
- ❑ D. `public void ejbPostCreate(Acct key) throws javax.ejb.CreateException`
- ❑ E. `public static void ejbPostCreate(Acct key) throws javax.ejb.CreateException`

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

7 Which method(s) from the EntityContext interface can be invoked from within the setEntityContext method? (Choose all that apply.)

- ❑ A. `getEJBObject()`
- ❑ B. `getEJBLocalHome()`
- ❑ C. `getCallerIdentity()`
- ❑ D. `getCallerPrincipal()`
- ❑ E. `setRollbackOnly()`

8 Which can be called on a CMP bean to transition it from the ready state to the pooled state? (Choose all that apply.)

- ❑ A. `ejbStore()`
- ❑ B. `ejbCreate()`
- ❑ C. `ejbSelect()`
- ❑ D. `ejbRemove()`
- ❑ E. `ejbPassivate()`

9 Which method(s) from the EntityContext interface can be invoked from within the ejbCreate method? (Choose all that apply.)

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `getCallerPrincipal()`
- ❑ D. `getUserTransaction()`
- ❑ E. `setRollbackOnly()`

10 What is true for a CMP bean in the ready state?

- ❑ A. Its `ejbLoad()` can be called directly after ejbStore.
- ❑ B. Its `ejbStore()` can be called directly after a business method.
- ❑ C. One of its business methods can be called directly after ejbStore.
- ❑ D. None of the above.

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

11 Which method(s) from the EntityContext interface must NOT be invoked from within the ejbLoad method? (Choose all that apply.)

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `getCallerPrincipal()`
- ❑ D. `getUserTransaction()`
- ❑ E. `setRollbackOnly()`

12 Which method, called on a CMP bean, is ALWAYS associated with a state change in the bean? (Choose all that apply.)

- ❑ A. `ejbLoad()`
- ❑ B. `ejbFind()`
- ❑ C. `ejbRemove()`
- ❑ D. `ejbActivate()`
- ❑ E. `unsetEntityContext()`

13 What's true about an CMP entity bean's primary key? (Choose all that apply.)

- ❑ A. The bean's primary key class must provide a suitable implementation of the hashCode and equals methods.
- ❑ B. When specifying the primary key in the deployment descriptor, only the field name must be declared
- ❑ C. All fields in the primary key class must be declared public.
- ❑ D. All fields used in the primary key must be container-managed fields.

14 How many ejbCreate methods can a CMP entity bean have?

- ❑ A. 0
- ❑ B. 1
- ❑ C. 0 or 1
- ❑ D. 0 to many
- ❑ E. 1 to many

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

15 Which method(s) are always invoked in direct response to a client operation? (Choose all that apply.)

- ❑ A. `ejbLoad()`
- ❑ B. `ejbCreate()`
- ❑ C. `ejbRemove()`
- ❑ D. `ejbActivate()`
- ❑ E. `ejbPassivate()`
- ❑ F. `setEntityContext()`

16 Which additional method(s) might the container call when invoking ejbRemove? (Choose all that apply.)

- ❑ A. `ejbFind()`
- ❑ B. `ejbLoad()`
- ❑ C. `ejbStore()`
- ❑ D. `ejbActivate()`
- ❑ E. `ejbPassivate()`

17 At what point(s) must the container establish a CMP bean's primary key? (Choose all that apply.)

- ❑ A. before calling `newInstance()`
- ❑ B. before calling `setEntityContext()`
- ❑ C. before calling `ejbCreate()`
- ❑ D. before calling `ejbPostCreate()`

18 Which method(s) run in the transaction context of the method that causes their invocation? (Choose all that apply.)

- ❑ A. `ejbLoad()`
- ❑ B. `ejbRemove()`
- ❑ C. `ejbSelect()`
- ❑ D. `ejbActivate()`
- ❑ E. `ejbPassivate()`
- ❑ F. `setEntityContext()`

Mock Exam Answers

1 What's true for a bean provider when creating an entity bean using container-managed persistence? (Choose all that apply.)

- ❑ A. Container-managed persistent fields must be defined in the entity bean class.
- ❑ B. Container-managed relationship fields must be defined in the entity bean class.
- ☑ C. When implementing a one-to-many relationship, the java.util.List interface must not be used.
- ❑ D. Accessor methods for container-managed relationship fields must be exposed in the bean's remote component interface.

– Only Collection or Set can be used

2 Which of the following is a legal accessor method for a persistent field in an entity bean with container-managed persistence? (Choose all that apply.)

- ❑ A. `public getCustomerNum();`
- ❑ B. `public void getCustomerNum();`
- ❑ C. `abstract void getCustomerNum();`
- ☑ D. `public abstract int getCustomerNum();`

– It must be "public abstract" and return the value of whatever the field is... it IS after all, a GETTER.

3 Which of the following are legal accessor method(s) in an entity bean with container-managed persistence? (Choose all that apply.)

- ❑ A. `public abstract int GetCustomerNum();`
- ❑ B. `public abstract int getcustomerNum();`
- ☑ C. `public abstract int getCustomerNum();`
- ❑ D. `public abstract int getCustomerNum() { };`

– you have to follow the Java naming convention

– not legal Java

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

mock exam answers

4 Which are requirements for a CMP entity bean class? (Choose all that apply.) (spec: 190)

- ❑ A. The class must define a **`finalize()`** method.
- ❑ B. The source file must define at least one constructor. – No, you want the default constructor
- ☑ C. The class must be declared public and abstract.
- ☑ D. The class must implement, directly or indirectly, the **`javax.ejb.EnterpriseBean`** interface. – EntityBean extends EnterpriseBean
- ☑ E. All getter and setter methods for the bean's abstract persistence schema must be abstract.

5 Which are legal declarations for a CMP bean's ejbCreate methods? (Choose all that apply.) (spec: 192)

- ❑ A. **`public void ejbCreateBigCustomer() throws javax.ejb.CreateException`**
- ☑ B. **`public String ejbCreateAccount() throws javax.ejb.CreateException`**
- ❑ C. **`static String ejbCreate() throws javax.ejb.CreateException`**
- ❑ D. **`public int ejbCreate() throws javax.ejb.CreateException`**
- ❑ E. **`public final String ejbCreate() javax.ejb.CreateException`**

– it can't be:
– static
– void
– final
must return the type of the primary key

6 Which are legal declarations for a method in a CMP bean? (Choose all that apply.) (spec: 192–193)

- ❑ A. **`public Account ejbSelectAcct(long x) throws javax.ejb.FinderException`**
- ☑ B. **`public abstract Acct ejbSelectAcct(long x) throws javax.ejb.FinderException`**
- ❑ C. **`public Account ejbPostCreate(Acct key) throws javax.ejb.CreateException`**
- ☑ D. **`public void ejbPostCreate(Acct key) throws javax.ejb.CreateException`**
- ❑ E. **`public static void ejbPostCreate(Acct key) throws javax.ejb.CreateException`**

– must be:
abstract
void
– can't be
static

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

7 Which method(s) from the EntityContext interface can be invoked from within the setEntityContext method? (Choose all that apply.) (spec: 179)

- ❑ A. **`getEJBObject()`**
- ☑ B. **`getEJBLocalHome()`**
- ❑ C. **`getCallerIdentity()`**
- ❑ D. **`getCallerPrincipal()`**
- ❑ E. **`setRollbackOnly()`**

– just too early for these last three. There's not a caller tx. setEntityContext just means a bean is going in the pool

8 Which can be called on a CMP bean to transition it from the ready state to the pooled state? (Choose all that apply.) (spec: 168–169)

- ❑ A. **`ejbStore()`**
- ❑ B. **`ejbCreate()`**
- ❑ C. **`ejbSelect()`**
- ☑ D. **`ejbRemove()`**
- ☑ E. **`ejbPassivate()`**

– store can be called anytime and is not called to transition the bean

– with ejbRemove(), the bean won't get an ejbPassivate()

9 Which method(s) from the EntityContext interface can be invoked from within the ejbCreate method? (Choose all that apply.) (spec: 179)

- ☑ A. **`getEJBHome()`**
- ☑ B. **`getEJBObject()`**
- ☑ C. **`getCallerPrincipal()`**
- ❑ D. **`getUserTransaction()`**
- ☑ E. **`setRollbackOnly()`**

– Entity beans can NEVER invoke this, because entity beans must use CMT

10 What is true for a CMP bean in the ready state? (spec: 169)

- ☑ A. Its **`ejbLoad()`** can be called directly after ejbStore.
- ☑ B. Its **`ejbStore()`** can be called directly after a business method.
- ☑ C. One of its business methods can be called directly after ejbStore.
- ❑ D. None of the above.

– The point is: load, store and business methods can be called by the Container in any order

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

mock exam answers

11 Which method(s) from the EntityContext interface must NOT be invoked from within the ejbLoad method? (Choose all that apply.) (spec: 180)

- ❑ A. `getEJBHome()`
- ❑ B. `getEJBObject()`
- ❑ C. `getCallerPrincipal()`
- ☑ D. `getUserTransaction()` – for BMT only
- ❑ E. `setRollbackOnly()`

12 Which method, called on a CMP bean, is ALWAYS associated with a state change in the bean? (Choose all that apply.) (spec: 168–169)

- ❑ A. `ejbLoad()` – not always!
- ❑ B. `ejbFind()`
- ☑ C. `ejbRemove()` – bean goes back to the pool
- ☑ D. `ejbActivate()` – bean comes out of the pool
- ☑ E. `unsetEntityContext()` – bean goes from the pool to death

13 What's true about an CMP entity bean's primary key? (Choose all that apply.) (spec: 203, 275)

- ☑ A. The bean's primary key class must provide a suitable implementation of the hashCode and equals methods.
- ❑ B. When specifying the primary key in the deployment descriptor, only the field name must be declared – no, you need the type
- ☑ C. All fields in the primary key class must be declared public.
- ☑ D. All fields used in the primary key must be container-managed fields.

14 How many ejbCreate methods can a CMP entity bean have? (spec: 171)

- ❑ A. 0
- ❑ B. 1
- ❑ C. 0 or 1
- ☑ D. 0 to many
- ❑ E. 1 to many

You don't have to let clients create beans

15 Which method(s) are always invoked in direct response to a client operation? (spec: 171–172)
(Choose all that apply.)

- ❑ A. **ejbLoad()**
- ☑ B. **ejbCreate()**
- ☑ C. **ejbRemove()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

Load, Passivate, and setEntityContext can be called by the Container when it wants to

16 Which additional method(s) might the container call when invoking ejbRemove? (Choose all that apply.) (spec: 176)

- ❑ A. **ejbFind()**
- ☑ B. **ejbLoad()**
- ❑ C. **ejbStore()**
- ☑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**

a bean has to be ready and loaded before it can do a remove – it may have cascading deletes to take care of

17 At what point(s) must the container establish a CMP bean's primary key? (spec: 175)
(Choose all that apply.)

- ❑ A. before calling **newInstance()**
- ❑ B. before calling **setEntityContext()**
- ❑ C. before calling **ejbCreate()**
- ☑ D. before calling **ejbPostCreate()**

the Container uses the values you set in ejbCreate so that in ejbPostCreate the bean is ready to use its key

18 Which method(s) run in the transaction context of the method that causes their invocation? (Choose all that apply.) (spec: 174–178)

- ☑ A. **ejbLoad()**
- ☑ B. **ejbRemove()**
- ☑ C. **ejbSelect()**
- ❑ D. **ejbActivate()**
- ❑ E. **ejbPassivate()**
- ❑ F. **setEntityContext()**

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