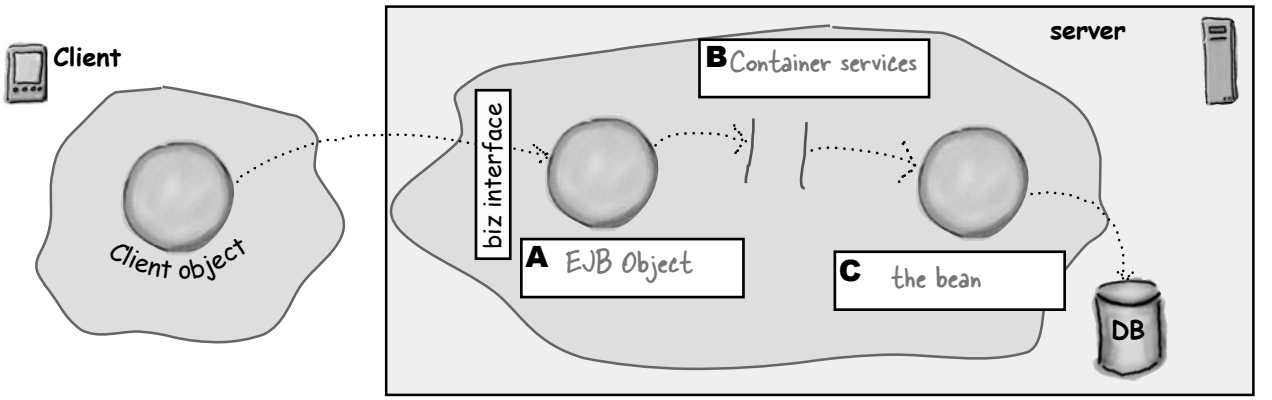
Sharpen your pencil

1. Label the three parts in the diagram.

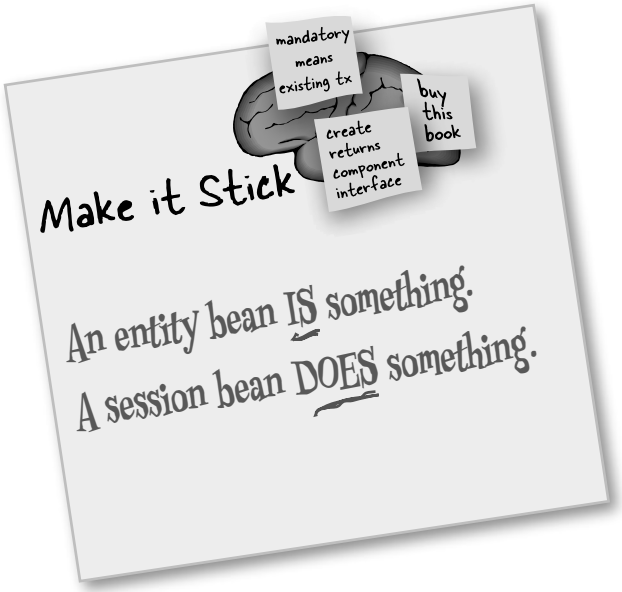


2. Describe (briefly) what each of the three things are responsible for, or how they behave.

A The EJB object is the bodyguard for the bean... it intercepts the calls coming from the client and passes the request to the Container.

B The Container services are the things you're paying for... the reason you're using EJB in the first place – transactions, security, persistence, etc.

C The bean has the real business logic. It has the actual functionality (method code) to do whatever it is this remote service is supposed to do (track your shopping cart, do a big calculate, update a customer's address, etc.



Sharpen your pencil

Know your bean types.

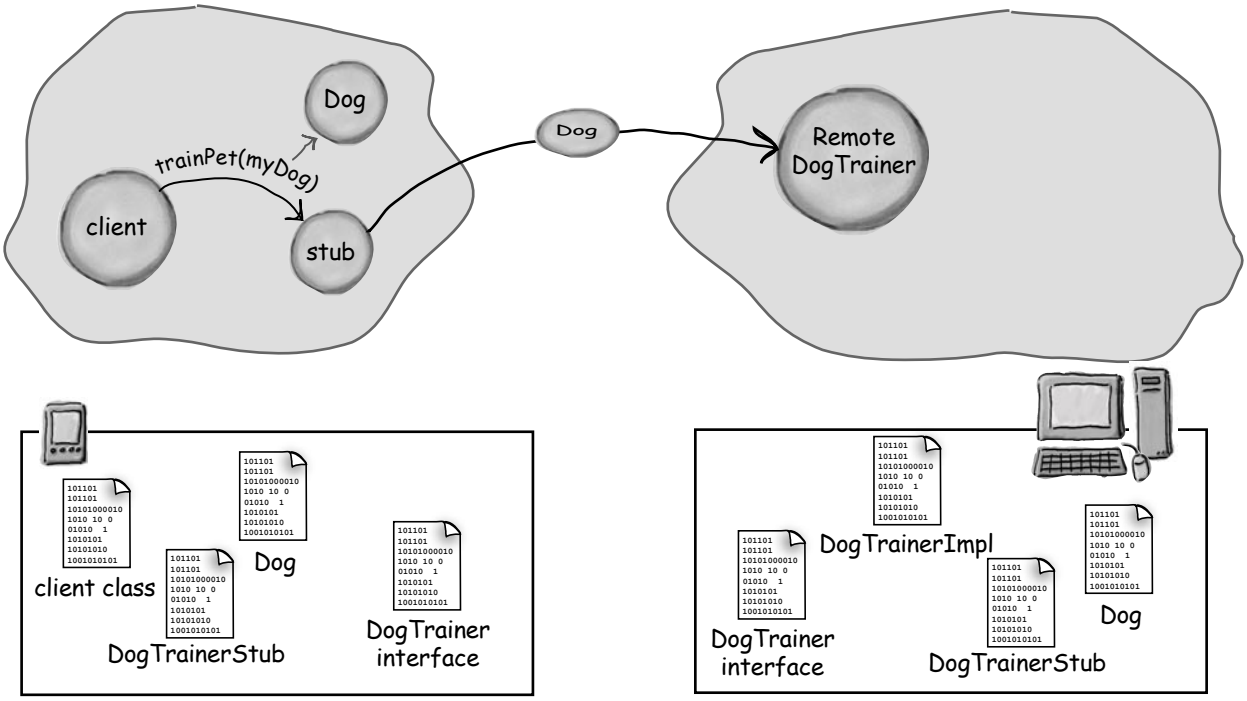
Look at the problem description on the left, and put a check mark for the bean type that would best fit the problem. There isn't one perfect right answer for these... you might decide that one bean type will work if you approach it one way, but another bean will work if you solve the problem in a different way.

	Entity	**Message-driven**	**Session bean** (circle stateless, stateful, or both)
Booking a ticket for a rock concert	☐	☐	☑ stateful stateless
			probably stateful because it might have to ask you several questions in order to complete the process
A bank account	☑	☐	☐ stateful stateless
Searching a product database	☐	☐	☑ stateful stateless
Dating service match-ups	☐	☐	☑ stateful stateless
Receiving submitted expense reports, and sending them out for approval	☐	☑	☐ stateful stateless
Online expert diagnosis—you describe a symptom and the system helps you determine the cause	☐	☐	☑ stateful stateless
The books in a library	☑	☐	☐ stateful stateless

Sharpen your pencil

This sharpen gets you on the one most common mistake EJB developers make. So don't skip it!

Based on this scenario, draw the classes below into the appropriate slot for whether they must be on the client, server, or both (you can reuse a class). The picture is simplified, so you aren't seeing all of the players involved.



the Dog class has to be on the Server as well as the client, because it must be deserialized when it gets to the server. Both the client and server need the interface (client won't even compile without the interface), but only the server needs the actual DogTrainerImpl (the thing that does the remote work on the server). The client needs the stub class, obviously, but the server usually will too, for plain old RMI, because it's the server that delivers the stub object to the client initially. (This isn't always true for RMI and EJB, but for now we'll pretend that it is.)

Sharpen your pencil

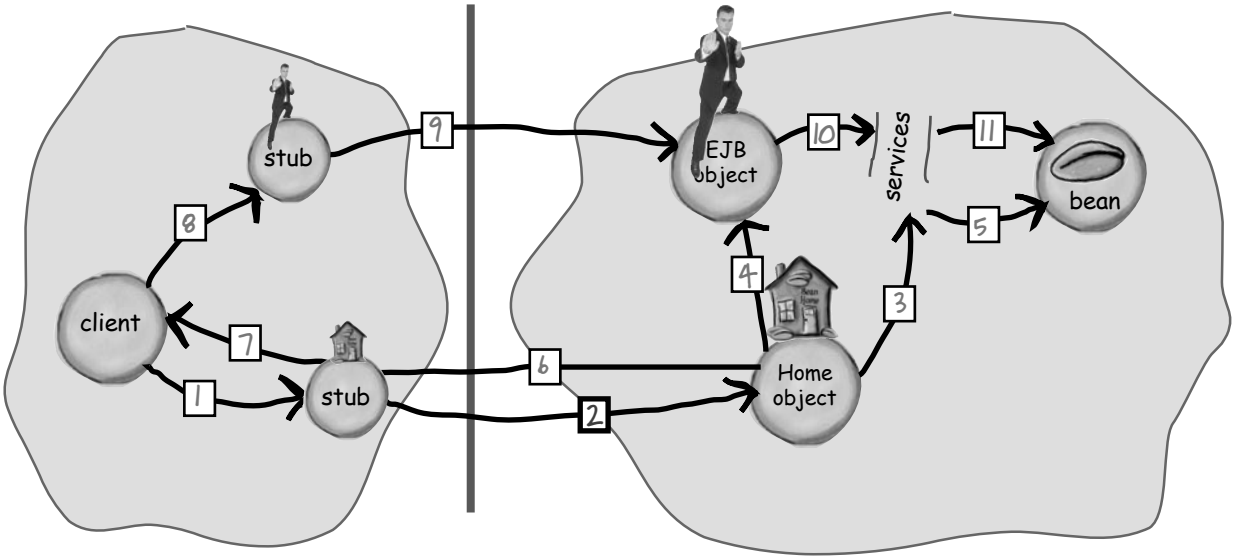
EJB Lifecycle:

Client has only a Home stub but wants to invoke a business method on a bean.

In the scenario below, assume the client has previously done a JNDI lookup and gotten back a stub to the Remote Home object.
Everything in the picture is what happens AFTER the client has the Home stub and now wants to get a reference to an EJBObject and ultimately call a business method on the bean.

Number the arrows (using the boxes over the arrows) in the order in which they occur. These arrows aren't necessarily direct method calls (although they might be), but rather arrows pointing to the next THING that happens. Tell a story for what happens at each arrow. There might be more than one right answer, depending on how you tell the story. Some arrows are missing; you can add them if you want, or, just assume some things are happening that you don't have arrows for.

Relax and take your time.
If you get stuck, flip back through the previous pages and study the diagrams.



1. client calls create() on the Home stub
2. The stub tells the Home that the client wants to "create" a bean
3. Home object interacts with the Container services (or you can think of the Home AS the Container)
4. EJB object is created for the bean
5. The bean is created (note: steps 4 and 5 can be in a different order depending on the bean type...)
6. Home returns the EJB object stub to the Home stub.
7. The stub is returned to the client
8. Client calls a business method on the EJB object stub
9. Stub contacts the EJB object with information about the method call (including any args)
10. Container services kick in...
11. Method call goes to the bean

Organize your beans

Finish the table by putting in a checkmark (even better if you add notes) in the boxes corresponding to the labels that apply to that bean type. We've done one of the boxes for you. If you get stuck, go back through the previous two chapters. You might have to make your best guess on a few things. That's OK—you'll have it all worked out way before the end of the book. We believe in you. You can do it. [cue theme song from "Rocky"]

	Stateless Session Beans	Stateful Session Beans	Entity Beans	Message-driven Beans
Uses a pool	Yes. Since they don't keep any client-specific data, you don't need one per each client.	Nope. Stateful beans never have a pool.	Yes. When an entity bean is not *being* a specific entity (row from DB), it stays in the pool	Yes. They're a lot like stateless session beans. Any bean of a particular type is the same as any other.
Multiple clients can have a reference to the same bean	No... although one bean can service multiple clients, but only one at a time.	Never! Would you want someone putting their shoes in YOUR shopping cart?	Yes -- there can be multiple clients accessing the same entity (like Fred #24)	Not applicable... message-driven beans don't have actual *clients*
Guaranteed to survive a container crash	No... although it's easy for the client to re-establish one that's identical to the one they had before the crash.	No.. You *might* have a vendor that provides this, but don't count on it.	Yes! The bean instance itself doesn't survive, but the underlying entity does.	No.
Has a client view	Yes	Yes		No.
Allows asynchronous communication	No.	No.	No.	Yes
Represents a process	Yes	Yes	No.	Yes
Represents a "thing" in an underlying persistent store (like a database)	No.	No.	Yes	No.

Let's take another look at the complete client code

```
import javax.naming.*;
import java.rmi.*;
import javax.rmi.*;
import headfirst.*;
import javax.ejb.*;

public class AdviceClient {

   public static void main(String[] args) {
      new AdviceClient().go();
   }

   public void go() {
      try {
         Context ic = new InitialContext();
         Object o = ic.lookup("Advisor");

         AdviceHome home = (AdviceHome) PortableRemoteObject.narrow(o, AdviceHome.class);

         Advice advisor = home.create();

         System.out.println(advisor.getAdvice());

      } catch (RemoteException rex) {
           rex.printStackTrace();
      } catch (CreateException cex) {
           cex.printStackTrace();
      } catch (Exception ex) {
           ex.printStackTrace();
      }
   }
}
```

A bunch of imports, we'll look at each one individually at the bottom of the page

InitialContext is our entry point into the JNDI naming service, where we do the lookup on the name "Advisor"

What is THIS??? Why not just a plain old cast?

Call create on the home to get us what we REALLY want — the component interface.

The point of everything! To call a business method on the bean (via the EJBObject stub)

Not a good way to handle (or rather, NOT handle) exceptions here... but we want to show some of the checked exceptions...

Sharpen your pencil

Match the class name with the package it's from. You can use the same package name more than once.

If you're not sure, make your best guess.

Package Name	Class Name
javax.naming	InitialContext
java.rmi	AdviceHome
javax.rmi	PortableRemoteObject
headfirst	RemoteException
javax.ejb	Advice
	CreateException

Sharpen your pencil

Based on the rules for session bean home interfaces, which statements are true about this interface:

```
import javax.ejb.EJBHome;
import java.rmi.RemoteException;

public interface CartHome extends EJBHome {

   public Cart create() throws CreateException, RemoteException;

}
```

- ☐ CartHome must not be the home of a stateful session bean.
- ☑ The interface is missing an import statement.
- ☐ The create method is missing an exception.
- ☐ Cart must be the class type of the bean.
- ☑ Cart must be the interface that extends EJBObject.
- ☐ The object returned from create() must be narrowed.
- ☑ The object returned from create() does not need a cast.

it can be either stateful or stateless

javax.ejb.CreateException

Cart must be the Remote component interface!

The object coming from create() already knows its type (Cart), so no cast or narrow is needed.

You MUST be able to look at code and infer a lot about it.

The exam expects you to look at client, interface, or bean code, and make inferences about things you don't see. You MUST know all of the rules for home and component interfaces. And there's more...

Sharpen your pencil

Based on what you now know about the difference between local and Remote client interfaces, decide if the following statements are true or false. You'll have to make some inferences and smart guesses for some of them.

Select all that are true:

- [x] The only way to remove a local session bean is through the component interface

because session beans don't have a primary key, and the only remove in a local home takes a primary key

- [x] Entity beans can be removed through a local home interface
- [ ] If you see an isIdentical() call, this must be a local bean

Handles always means Remote!! They aren't needed for local clients...

- [x] If you see a getHandle() call, this must be a Remote bean
- [ ] If the client is catching a RemoteException on a home method, the bean's home interface must extend EJBLocalHome

RemoteException is REQUIRED for remote interface methods, but you must NOT use them with local interfaces.

- [ ] If the client is not handling a RemoteException on a business method, the bean's component interface must extend EJBObject.
- [ ] If you see a call to getEJBMetaData(), the bean's component interface must extend EJBLocalObject.

EJBMetaData is just for Remote clients

- [ ] If you do a JNDI lookup on a local home, you must narrow the object returned from JNDI

You'll have to cast it, but not narrow it. Narrowing is just for Remote stubs.

- [ ] There are three methods in the EJBLocalObject interface

Turn back to page 154 to see the interfaces...

- [ ] There are two methods in the EJBLocalHome interface