Enterprise JavaBeans 3.0

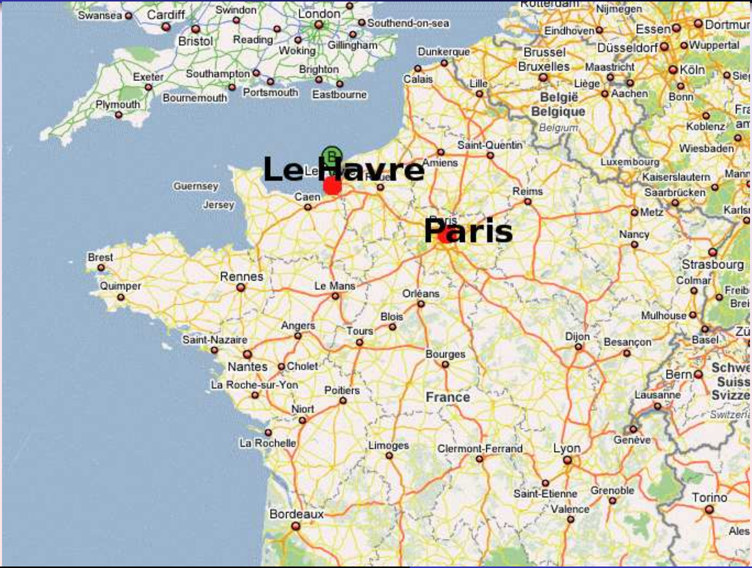
Claude Duvallet

University of Le Havre
Faculty of Sciences and Technology
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX, FRANCE
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/index-en.php

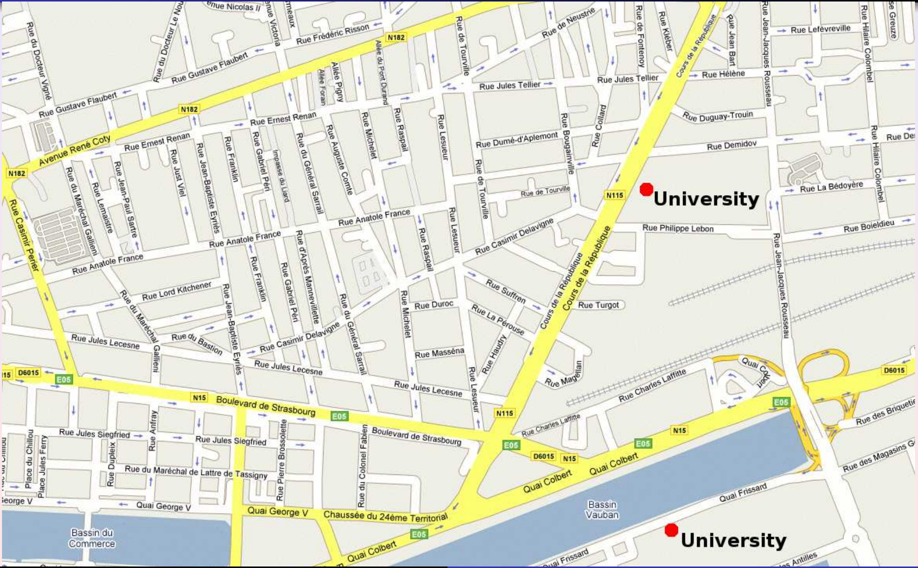
Who am I?

- Wǒ tiaò Duvallet Claude, Wǒ shì fǎ guó rén
  - Associate Professor in computer science since September 2003.
  - PhD obtained in October 2001 at Le Havre University, France
- Where do I come from? Le Havre University (France) dà xué lè ā fuó ěr.
- My topics of interest:
  - Teachings: Programming (Java, C/C++), Operating Systems (Linux, Unix), Distributed System (CORBA, RMI, RPC, EJB, LDAP), Network Protocols and Architectures, Network and System Administration.
  - Research: Real-Time Databases, Multimedia Systems, Quality of Service Management, Distributed Systems, etc.
- Current PhD supervising: Nizar Idoudi, Emna Bouazizi and Bechir Alaya.
- My homepage in English: `http://litis.univ-lehavre.fr/~duvallet/index-en.php`

Le Havre in France



University of Le Havre



Presentation of the University of Le Havre

- University of Le Havre is a small university: 7000 students.
- Four topics of studies:
  - Sciences, Technologies, Health.
  - Law, Economics, Management.
  - Letter Language.
  - Social and Human Sciences.
- 3 Faculties and 2 Institutes:
  - Faculty of Sciences and Technologies.
  - Faculty of International Affairs.
  - Faculty of Letters and Humanities.
  - Institute of Technology.
  - Institute of Logistic.

Computer Science,
Information Processing, and
Systems Laboratory

Enterprise JavaBeans 3.0

Part I: Introduction to the Enterprise JavaBeans

Introduction to the Enterprise JavaBeans

Architecture of an information system

To take into account:

- access to the data,
- treatment of the data,
- presentation of data,
- network connectivity,
- transaction design,
- application security.

Architecture in layer (1/3)

- It allows to master the design of an information system, as well as its evolution by dividing the roles in the forms of software layers.
- A classical architecture is a 3 layers architecture:
  - the presentation layer,
  - the business layer,
  - the data layer.
- The presentation layer:
  - it contains components which realize the computer human interface of the application and manage the interactions with the user.
  - it can be developed with Motif, MFC or JFC (swing), or yet a Java applet, an ActiveX or a dynamic or static HTML page.

Architecture in layer (2/3)

- The business layer:
  - It contains components which realize the management of business rules.
  - Example: for a bank applications, the components may be object that models account, contract,... The object are written in C, C++, Java...
- The data layer:
  - It is used by the business layer to store the states of the objects in a persistence support (relational or object database...).
  - The data layer may change or evolve independently from the other layers.

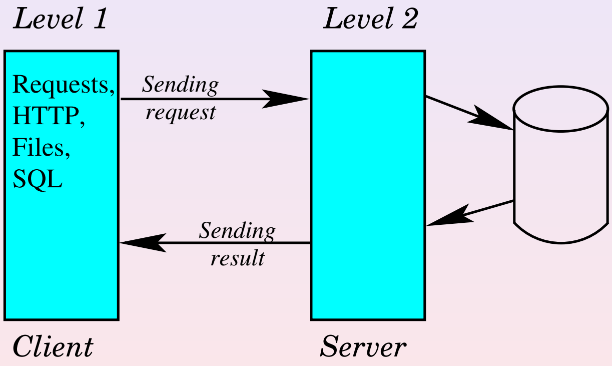
Architecture in layer (3/3)

- The main objective of the layers separation:
  - to insure the Independence of the business logic from the problems of visualization an storage.
  - $\Rightarrow$ A more modular application, better structured and easier to maintains.
- We can envisage intermediate layers between the precedent layers:
  - an applicative layer applicative between the presentation and business layers (controllers...),
  - a layer of technical services between the business and data layers.

Architecture client/server (1/2)

- The physical separation of the layers is an other problems.
- In a two layers architecture, two layers are grouped and so separated from the third one.
- Particularly, it is the case in the client/server applications where the treatment and the display are done on the client computer whereas the data are stored on a server. They are called "heavy clients" with a high cost to deploy it.
- An other method consist on grouping together the business logic and the storage of the data on the server.

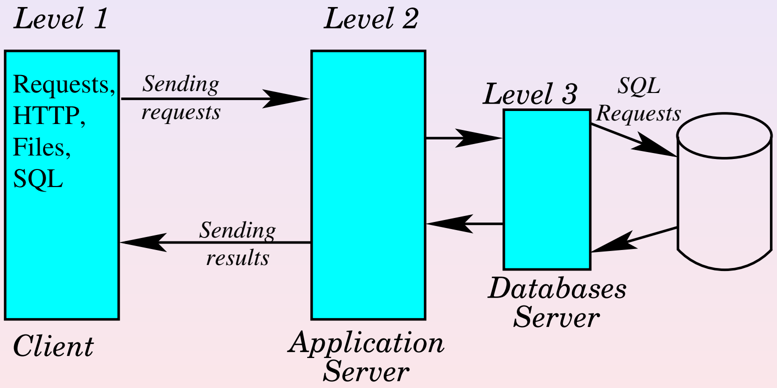
Architecture client/server (2/2)



The 3-layers architecture (1/3)

- The 3 layers may be completely physically separated:
  - It is possible to decompose all these layer in sub-layers located on different computer or hosts.
  - So, we talk about n level architecture.
- It is the case of the WEB application:
  - the presentation layer is constituted of JSP pages and servlets,
  - the business layer is implemented thanks to some beans or object hosted by an application server,
  - an the data layer consist on one or many database.
- Interest of the 3 levels architecture:
  - better support the increases of load,
  - the share of the connection to the resources,
  - easy to deploy, security.

The 3-layers architecture (2/3)



The 3-layers architecture (3/3)

- The server becomes the hearth of the system.
- All the logic of the application is implemented in some components hosted by this server called application server.
- It provides to them a runtime environment but also a set of services.
- The business logic may be provided by many applications server that communicated together. The business objects are distributed.
- For example, in an e-commerce application, the computation needed to make accounting treatment may be supported by components located on a first server whereas the components which manage the stocks are located on an other server.

Middlewares

- The distributed objects need a technical framework in order to allow the communication between these objects.
- This framework must provide to the objects an access at many services like naming service, transaction management, persistence management, security management...
- In transparent way for the developer that is to say without he needs to implement the mechanism to insure these services.

⇒ This kind of framework is called Middleware.

- Other services are needed:
  - optimization of the access to the resources (pool of connections...),
  - mechanism to activate/passivate the objects,
  - mechanism to distribute the load and to manage fault tolerance.

Applications servers

- Unfortunately, they were not interoperable because of the lack of standard specifying the Middleware to use, the provided services, the interfaces to access to them...
- The J2EE architecture is a standard proposed by Sun for the JAVA applications servers.
- At the center of this architecture, we find a standardized Middleware, based on RMI/IIOP, and business objects which are distributed Java object called EJB (Enterprise JavaBeans).
- An application server is a complet environment, which contains the EJB container and the Web server.
- To insure the portability, Sun a specified a set of compatibility which consist nearly 6000 tests that the server must satisfy to have the SUN certification.

J2EE architecture

- The EJB technology is included in a more large platform called J2EE.
- This platform consist on an architecture for the development, the deployment and the execution of the distributed application.
- These applications require some technical services like transactions management, security management, the access by the client, the database access.
- The J2EE platform provides all theses technical services.
- The developer could focus on the business logic instead of to disperse on many technical problems.

J2EE technologies (1/3)

The included technologies in J2EE, provided thanks to some API, allow:

- The communication between distributed objects with RMI (Remote Method Invocation) and RMI/IIOP protocol.
- The creation of transactional distributed objects with some EJB (Enterprise JavaBeans).
- The research and the getting back, in a server, of the reference names on distant objects with JNDI (Java Naming and Directory Interface)
- The access to the databases with JDBC (Java DataBase Connectivity)
- The transactions management with JTA (Java Transaction API) and JTS (Java Transaction Service).

J2EE technologies (2/3)

They also allow:

- The asynchronous communication by some messages between the distributed objects with JMS (Java Message Service).
- The realization of WEB graphical interfaces with the JSP pages (JavaServer Pages) and the servlets.
- The integration of CORBA objects thanks to JavaIDL.
- The electronic mails sending thanks to JavaMail.
- The integration of existing systems thanks to the connectors.
- The descriptions of behavior of the Java components in XML.

J2EE technologies (3/3)

- The business logic is implemented in the EJB which are transactional components on side of the server and accessible at distance by the clients.
- The EJB are executed in J2EE servers that works as intermediates servers in a client/servers systems.
- Two kinds of connectors are used: the Web container and the EJB container.
  - The Web container is an execution environment for the JSP pages and the servlets which are a gateway between the user interface and the EJB implementing the business logic.
  - The EJB container is an execution environment for the EJB. The EJB container hosts the business component and provides them some services.

The EJB container

- From the server side, the access to the different resources is complex.
- With the EJB, most of the tasks are done by declaration, avoiding to write specific code to manage the transactions or the persistence.
- It allows to focus on the development of the component by delegating to the EJB container the implementation of the technical services and the providing of these services during the execution.
- At each component is associated a deployment descriptor, to specify, for example, that a bean is persistent, secured and accessible by many client in the same time.
- The server is in charge of the treatment of this descriptor and insure the execution of theses services.

Main functionality of an EJB container (1/2)

The main functionality provided by an EJB container are:

- The connectivity between the clients and the EJB:
  - the container manages the communications between the client and the EJB.
  - after the deployment of an EJB in an applications server, the client may invoke the methods as it were located in the same virtual machine.
  - the communications are insured by the Middleware in a transparent manner.
- The management of the persistence:
  - the persistent components may choose to delegate the persistence to the container.
- The management of the transactions:
  - the persistent components may choose to delegate the management of their transactions to the container, which implements the transactional mechanisms needed for the realization of the transactional logic describe by the beans.

Main functionality of an EJB container (2/2)

- The management of the security:
  - security policy declared but not implemented by the developer,
  - security management base on the security API of Java,
  - methods links to the security implemented by the container,
  - utilization of the security attribute define in the deployment descriptor of the bean used during the phase of deployment.
- The concurrency management:
  - The components are invoked by only one client by many clients simultaneously.
- The management of the life cycle of the components:
  - creation and destruction of the instances of the components.
- The management of a pool of connections:
  - connection to a database = costly in terms of resources,
  - number of connections are limited by the number licenses,
  - solution: the container manage a pool of connections.

The Enterprise JavaBeans (1/3)

- The development of software has seen appear in the 90's years the notion of component as some pieces of standard code, reusable and which encapsulate the business logic.
- The EJB components are designed to encapsulate the business logic and avoid the the developer to be worried about all is around the system: transaction, security, concurrency, communication, persistence, errors managements...
- An EJB component consist on a collection of JAVA class and a XML file, merged into an unique entity.
- The container take into charge all concerning the systems. This separation of the tasks is the fundamental concept of this technology.

The Enterprise JavaBeans (2/3)

- An EJB component is designed as a reusable set of business logic and works with all kinds of clients: servlets, JSP, Java/RMI applications,...
- The specification Enterprise JavaBeans 1.1 define 2 types of beans: Entity Bean and Session Bean.
- La specification Enterprise JavaBeans 2.0 introduce a third bean: the Message Driven Bean. So it exists three types of beans:
    - the Session Beans,
    - the Entity Beans,
    - the Message Driven Beans.

The Enterprise JavaBeans (3/3)

```
       Enterprise JavaBeans
               |
    --------------------------------
    |                 |            |
 SESSION           ENTITY       MESSAGE
    |                 |
 ----------        -------
 |        |        |     |
with   without    CMP   BMP
state   state
```

The Session Beans (1/2)

- A Session Bean represents a process. It is an extension of the client process into the J2EE application server.
- We distinguish 2 types of Session Beans: the Session Beans without a state (stateless session) and the Session Beans with a state (statefull session).
- A Session Bean have only one client at one given moment.
- For a Session Bean without a state, many clients may be associated at the same bean successively.
- For a Session Bean with a state, it is the same client that's making all the invocations.
- A Session Bean without a state represents a functional treatment, like the calculation of a route between two points of the demand of a transfer between two bank accounts.

The Session Beans (2/2)

- A Session Bean with state have a conversational state, that is to say a state resulting from the interaction with the client.
- Example of a Session Bean with state:
  - a basket on a site of electronic commerce which have two attributes: one for the name of the customer and one for the articles selected by this customer.
  - This bean may have a method lit `addArticle()` that the customer/client may invoke to add a new article in the basket.
- The state represented by a Session Bean is private and conversational. The bean is accessible by only one unique client.

The Entity Beans (1/2)

- An Entity Bean represents a persistent business object.
- For example, an Entity Bean represents a command, an article, a bank account.
- The attributes of an Entity Bean can be stored in a database or any other means of persistence. An Entity Bean can be used by several clients simultaneously.
- The persistence of an Entity Bean may be managed by the bean itself or by the container.
  - In the first case, the operations of reading and writing on the support of persistence must be encoded in the bean (for example, SQL code for a relational database).

  $\Rightarrow$ we are talking about Bean BMP (Bean Managed Persistence).

The Entity Beans (2/2)

- In the second case, persistence is managed by the container and only the functionality of advanced research must be encoded in the bean, other operations of reading and writing on the support of persistence are carried out automatically by the container.

  $\Rightarrow$ we are talking about Bean CMP (Container Managed Persistence).

- The state represented by an Entity Bean is shared and transactional. The bean is the single point of access to such data by different clients.

The Message Driven Beans

- The Session Beans and Entity Beans are components distributed invoked by clients synchronously, that is to say by invocation of methods,
- The Message Driven Beans are beans which consume messages asynchronously, through the Java Message Service (JMS).

EJB Interface

- EJB components have EJB interfaces exposing their services available to customers.
- Customers use these interfaces to execute the logic encapsulated in the bean.
- There are 2 kinds of interfaces: the Home interface and the business interface.
- The Home interface is used to create, delete instances of the bean.
- The business interface is used to execute the business methods included into the bean.

EJB: some distributed objects (1/3)

- distributed objects are useful because they allow strengths to distribute an application on a network.
- However, requirements such as transactions and security are becoming indispensable in business applications.
- A distributed object is an object that can be called from a remote system, including:
  - from a customer as part of the process containing the object (in-process),
  - from a customer outside this process (out-of-process),
  - or from a client located anywhere on the network.

EJB: some distributed objects (2/3)

- The means of call for a distributed object are:
  1. The client call a stub, which is a client-side proxy object.
     - The stub mask the client communication network.
     - The stub sends calls over the network through sockets, by manipulating the parameters as appropriate in their network representation.
  2. The stub called a skeleton on the network, which is an object proxy server side.
     - The skeleton mask made communication network level to be distributed.
     - The skeleton is able to receive calls over a socket and manipulate the settings in their representation network.
     - The skeleton delegates the call to the object distributed.
  3. The distributed object does its task and returns control to the skeleton, which sends it to the stub for what the latter returns to the customer.

EJB: some distributed objects (3/3)

- The stub and distributed object implement the same interface, called remote interface.
    - A customer who calls a method of distributed object calls in fact a method stub.
    - We talk about transparency local/remote.
- Different technologies can use objects such as distributed by OMG CORBA, DCOM Microsoft and Java RMI-IIOP Sun.

Example of J2EE application

A typical example of application based on J2EE architecture is that an application of e-commerce:

- The client connects to the Web site of a store, consult a catalogue of items available, it chooses some places in a shopping cart and adjusts its purchases.
- On the side of the Web server, JSP pages and servlets use:
    - some Entity Beans that reflect the business objects such as articles, customers, orders, invoices and Session Beans with state representative baskets customers.
    - some Session Beans without state which can implement treatments such as web browsing in a catalogue or request a quote for the contents of a basket.

Part II: Session Beans

Sessions Beans

Introduction

- A Session Bean is a component providing a business process.
- It implements the business logic, business rules, such as processing an order, the application processing of bank,...
- This is the extension of the client process in a J2EE application server.
- The client can be a Java application independently, an applet, a servlet or another Session Bean or entity. It seeks the services of the bean through its business interface.
- Two types of Session Beans: with or without state.

Session Bean without state

- A stateless Session Bean is a collection of services, each represented by a method.
- The bean does not preserve a state of a call to another.
- When one invokes a method on a bean without state, it executes the method and returns the result; he was not concerned about what method was used to be invoked before or after.
- The client keeps a reference on the subject EJB but the bean is free to serve invocations of methods from other objects EJB.
- The customer sees the bean through the EJB object which has a lifespan related to the user session: the instance of the class implementation can exist before and after the creation of the EJB object.
- Examples of conventional Session Beans without state services calculations, information retrieval in a comic ...

Session Bean with a state

- A stateful Session Bean is an extension of client server application.
- It conducts operations on behalf of a client and maintains a clean complementary to the client.
- This state is shown by many instances variables of the bean and is kept between the various calls made by the customer during its conversation with the bean.
- It's called a conversational state.
- It can be read or altered by the methods of the bean.
- The classic example is the eCart (electronic basket).

Implementation class of a Session Bean (1/2)

- The class implementation of a Session Bean implements the SessionBean interface. This interface inherits the interface EnterpriseBean.
- A Session Bean can support one or both business interfaces and one or both interfaces Home as the bean will be accessible at a distance and/or locally.
- The class implementation does not declare that it implements these interfaces, but must have methods that correspond to the methods of business and Home interfaces!
- This class must be public, not abstract and not final.
- It must have a public constructor without argument and should not redefine the method finalize ().

Implementation class of a Session Bean (2/2)

- A Session Bean without a state must implement the creating method `ejbCreate ()` without any argument whereas a stateful Session Bean must implement a creating method `ejbCreateXxxx ()`.
- The methods of creation must be public, not static, not final.
- A creating method throw the exception `CreateException` if one of the inputs is not valid.
- In the case of a remote interface, the arguments and return value must be compatible with RMI.
- The methods of the interface SessionBean defined by the class implementation will be invoked by the container to inform the bean of its course in its life cycle.

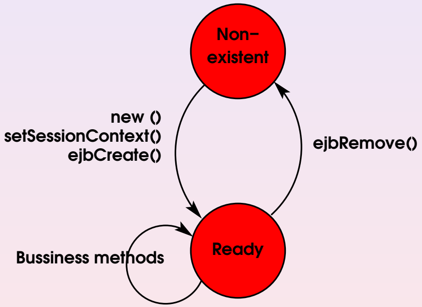
Lifecycle of a bean

- An application server is likely to mount a charge, that is to say to serve a large number of clients with limited resources.
- It will load into memory a limited number of beans.
- Some will be disabled (passivation) and swapped and vice versa (activation).
- The server informs the bean if it should turn it off so that it frees resources acquired before moving to the passive state.
- Same thing for the reactivation to restore resources. It is the role of the methods `ejbActivate ()` and `ejbPassivate ()` which are invoked by the container.
- The Session Beans have a different life cycle as the case (without state and with state).

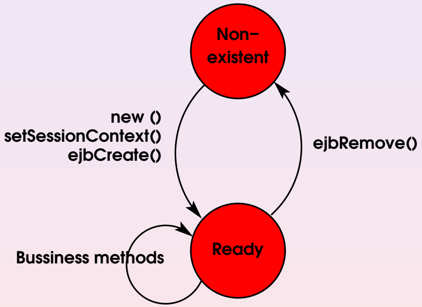
Lifecycle of a Session Bean without state (1/2)

- A single bean can be used by the container to serve requests from different customers, a query at a time, the beans are not multithreaded.
- The container creates a number of beans he puts in a pool and they are distributed on the various motions.
- It may possibly create new instances if the load increases. Or remove for the opposite reasons.
- There was no concept of activation or passivation: methods ejbPassivate () and ejbActivate () must be empty.
- A stateless Session Bean is never passivated, its lifetime is reduced to two states: non-existent and ready to receive invocations.

Lifecycle of a Session Bean without state (2/2)



Lifecycle of a Session Bean without state (2/2)



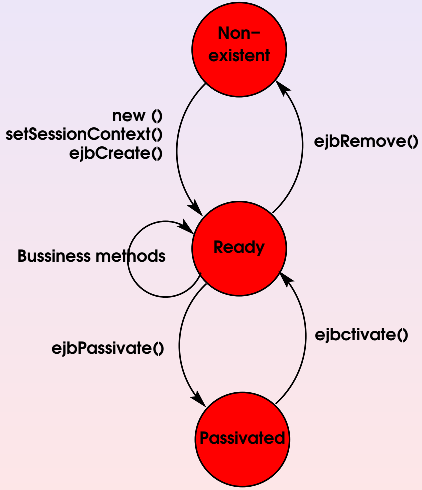
The call to the `create()` method provides a reference on an object EJB.

On appeal to a business method, an instance is selected to serve the request.

Lifecycle of a Session Bean with state (1/4)

- A stateful Session Bean is associated with one customer. It is activated or passivated by the container.
- At a passivation, the container serialize the bean to be saved.
    - In fact, only serializable variables of instances and the references of objects in the container (SessionContext) are stored.
    - Other variables of instances are loaded to the program (closing a connection and then restored when the activation).
- `ejbActivate()` is almost identical to `ejbCreate()`: connections to the acquisition of necessary resources.
- `ejbPassivate()` is almost identical to `ejbRemove()`: liberation of the connections to resources.

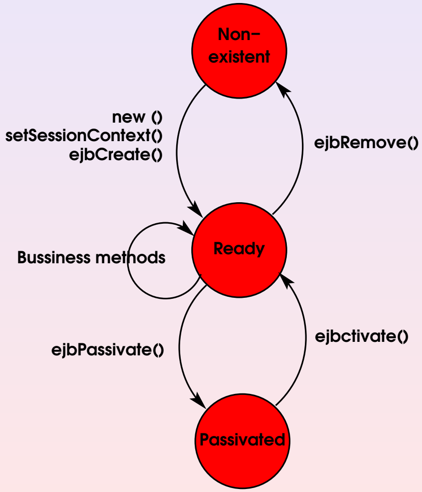
Lifecycle of a Session Bean with state (2/4)



Lifecycle initialization:

- The client initiates the life cycle by invoking `create()`.
- The container creates an instance of the bean and invokes methods `setSessionContext()` and `ejbCreate()`.
- Then the bean is ready to receive calls of business methods of the client.

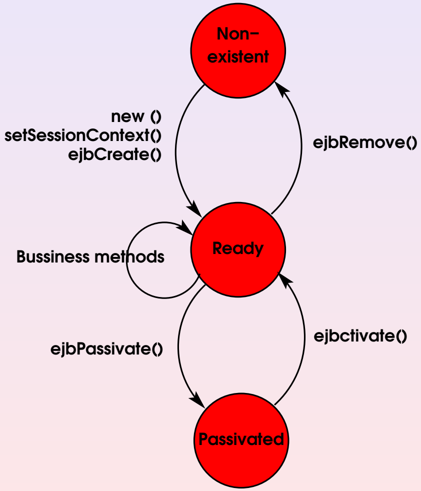
Lifecycle of a Session Bean with state (3/4)



Deactivate the bean:

- When the bean is in the ready state, the container can disable or passivate the bean by deleting it from memory to a secondary memory (for example, the bean the least used).
- The container invokes ejbPassivate () just before passivate the bean.
- If the client invokes a method of bean processing in the state, the container activates the recharging in the method then invokes `ejbActivate ()`.

Lifecycle of a Session Bean with state (4/4)



Ends of the lifecycle:

- when the client invokes `remove ()`.
- The container invokes then `ejbRemove ()` and dereference the bean that can be deleted by the garbage collector.

Control of the life cycle:

- The developer does not control the life cycle on methods `createXxxx()`, `remove()` and the business methods.
- All other methods are invoked by the container.

Collaboration between beans

- An application based on EJB involves a number of beans that will work together.
- Each bean has a well-defined, but it may request services from other beans.
- The following example illustrates how such a simple collaboration.
- It is a combination of service for Web client.
- Whenever the customer provides a value, it is added to the current content of an accumulator dedicated to the customer.
- This accumulator will in turn use the services of a bean Calc to make the bill.

Part III: Entity Bean

Introduction (1/3)

- An Entity Bean can implement objects or components modeling a reality business with data and not just services.
  - For example, accounts or orders are business objects.
  - These objects have an existence independent of processes that use them: the account continues to exist between two banking.
  - The Entity Bean is a complete implementation of the concept: the bean that models account will have as many instances as the bank accounts.
  - Each instances contains data associated with the account with its own identity. At a given moment, all the instances are not in memory.
  - The container instantiates a timely bean recovering the data stored in a database.

Introduction (2/3)

- When there is an update, the container has the responsibility to save the data in the database.
- In case of simultaneous access to the same bean, the container manages the competition for access directly at the Entity Bean.
- Any use of the business object, as the account will be through the bean.
- The server provides the same guarantees in respect of transactions, security, ...
- Like centralized databases have made it possible to reuse data across multiple applications, beans allow reuse of data and logic associated among several applications.

Introduction (3/3)

- Moreover, the appearance and spread of standardized EJB mask the heterogeneity of different databases or sub-systems.
- The EJB is a structural element in the development of applications.
- A J2EE application implements the separation of layers:
  - display level (standalone application, servlets, JSP),
  - application level (the calling program and stateful Session Bean),
  - business level (Entity Beans or Sessions Beans without state).

Mapping between bean and database

- Mapping between a bean and a database consist on knowing how attributes of the bean will be recorded in the database.
- At the simplest, the attributes of a bean correspond to the attributes of a tuple in a table.

Types of Entity Beans

- The persistence of data is managed in 2 ways.
- Synchronizing with the support (the base) can be managed by the container (Container Managed Persistence: CMP) or by the bean (Bean Managed Persistence: BMP).
- In the first case, it can declare the characteristics of access to the base to be borne by the container.
- In the second case, it will encode the different treatment needed to communicate with the support of persistence directly into the bean (JDBC).
- Synchronizing with the support (research, update) is done at the container to the mapping in a transparent manner.
- The user of bean do not know if it works CMP or BMP. Nevertheless from the development point of view, the choice is important.

Part IV: Messages Driven Bean

What is it a message driven bean?

- They were introduced in the specification EJB 2.0 to take into account the processing of asynchronous messages.
- They enable applications to treat J2EE (mainly) messages Java Message Service (JMS), and also other types of messages, while behaving as an earpiece (listener) JMS message it treats asynchronously.
- The Message Driven Bean implements the javax.jms.MessageListener interface that allows it to respond to messages received via the JMS method `onMessage()`.

Some similarities with the stateless Session Beans

- They keep no state, no data of a specific client.
- All instances of Message Driven Beans are equivalent.
- The EJB container can send messages to any of them, and these messages can be processed through competition and a pool of instances managed by the container.
- A Message Driven Bean can send messages to many clients.

Characteristics of the Message Driven Beans

- They run at the receipt of a message from a client.
- They have a life rather short.
- They are invoked asynchronously.
- They are not directly shared data in a database, but can access and modify them.
- They are without state.
- They can not be carried out in an existing transaction (EJB specification 2.0), but can create a new transaction in the method onMessage (), during which several operations can be carried out.

Some differences with the Session Beans without state

- The client does not use interface to access the Message Driven Bean.
- The client consists on a single bean (class).
- The use of Message Driven Beans is preferred to the use of Session Beans without state when messages must be received asynchronously so as not to saturate resources on the server side.

API JMS (Java Message Service) - Generality (1/2)

- JMS API is not exactly an API but a specification for the Sun management messages of the J2EE platform.
- It is composed mainly of interfaces that simplify the work of developer for sending or receiving messages.
- Four actors are involved in the use of JMS in a system:
    - The message itself, which is transmitted between 2 applications.
    - The applications that provide and receive the message, for example, a customer outside the application server and EJB application or J2EE.
    - The applications must be written in Java.

API JMS (Java Message Service) - Generality (2/2)

- A JMS provider that provides administration tools for the messages and who is found responsible for the messages that are sent to the issue.
- The objects which are administered resources to be found in the application server or a directory of JNDI provider. There are 2 kinds of objects:
    - the ConnectionFactory which are "factories connection" used to connect to the provider,
    - and "Destination" who are either "Queue" is "Topic".

API JMS (Java Message Service) - Two modes of connection

- The mode **Point to Point** in which it was only one recipient for each message .
    - The message is sent to a queue (tail) and is removed from the JMS provider as soon as it was "consumed" (or has expired).
    - The producer and consumer of the message need not be connected at the same time.
- The mode **Publication/Subscription** (Publish/Subscribe) is similar to how a news server.
    - A message was posted on a "Topic" and the consumers of this message must register the Topics of interest to them.
    - The message is deleted from the provider when it was read and paid by all subscribers Topic. It may therefore have multiple recipients.

API JMS (Java Message Service) - Two modes of consumption

- Two consumption of messages from a Queue or a Topic can be received in 2 ways:
    - Synchronously, the application is blocked when calling the method receive (). An overloaded version of this method can make the hand after the expiry of a timeout.
    - Asynchronously, the application is informed of the arrival of a message via a `MessageListener`. This thread is launching a listener which attends messages and executes a method at their arrivals.

The norm 3.0

Part III: The norm 3.0

Introduction

- The standard EJB 3.0 is an important development in the field of EJB.
- It is an important part of the standard Java 2 EE 5.
- A new management model data persistence inspired Hibernate Jboss.
- A simplification very important developments.
  - Using annotations introduced in version 5.0 of Java.
  - Adoption of POJO programming model.
  - Removing deployment descriptors.
  - Removing of business interfaces and interfaces Home.
- The standard EJB 3.0 also incorporates an Aspect Oriented Programming (AOP) by introducing the concept of interceptor (Interceptor).

Objectives of the norm 3.0

- Simplification of the definition of interfaces, eliminating a number of pre-requisites in the standard 2.1 (no legacy of super classes or interfaces).
- Simplifying the creation of Bean.
- Simplification of APIs for access to the environment of the Bean: defining a simple injection dependent.
- Introduction annotations in Java instead of the deployment descriptor.
- Simplification on the persistence object facilitated by the use of object / relational mapping based on the direct use of Java classes and non-persistent components.

The annotations

- The annotations metadata allowing some tools to generate additional constructions in the compilation or execution.
- They simplify writing programs.
- They can happen deployment descriptor. Nevertheless, it is always possible to use it.
- It can specify interfaces local or remote, with only Bean keywords "@Remote" and "@Local."
- It can define a session Bean with or without state with only the keywords "@Statefull" or "@Stateless."

Hello World with the EJB 3.0 (1/4)

- Definition of the interface Hello:
  - Now, this interface is the only one to define.
  - Interface extremely simple: no need to have to any inheritance.

```
package hello;

public interface Hello {
    public String hello (String msg);
}
```

Hello World with the EJB 3.0 (2/4)

- Definition of the Bean class:
    - This class is also very simplified.
    - It does not inherit specific classes.
    - It must overcome the abolition of local and remote interfaces by using annotations.
    - We use an annotation to specify that it is a stateless Session Bean.

```
package hello;

import javax.ejb.*;

@Stateless
@Remote (Hello.class)
public class HelloBean implements Hello {
    public String hello (String msg){
        System.out.println ("Message received: "+msg);
        return "Hello "+msg;
    }
}
```

Hello World with the EJB 3.0 (3/4)

- Definition of deployment descriptor: it has also almost nothing that simplify its writing.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ejb-jar xmlns="http://java.sun.com/xmlns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                             http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd"
         version="3.0">
  <enterprise-beans>
  </enterprise-beans>
</ejb-jar>
```

Hello World with the EJB 3.0 (4/4)

- Definition of the client: it is also simplified compared with the same score in the EJB 2.1.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import hello.*;
import java.util.*;

public class HelloClient{
    public static void main(String[] args) throws Exception {
        Properties props = System.getProperties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
                  "org.jnp.interfaces.NamingContextFactory");
        props.put(Context.URL_PKG_PREFIXES,
                  "org.jboss.naming:org.jnp.interfaces");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        Context ctx = new InitialContext(props);
        Hello hello = (Hello) ctx.lookup("HelloBean/remote");

        System.out.println(hello.hello("World"));
    }
}
```

Lifecycle of the Beans 3.0 (1/3)

- Loss of Home interface: references on the beans are done directly.
- Using methods "callback" for the life-cycle management.
- The callback methods can be defined:
  - either directly in the bean: the signature of the method must be public void method_name (),
  - or in a class Listener Callback which resembles a home interface.
- The class Listener Callback:
  - all methods are defined in the class.
  - the methods to take the argument corresponding bean.
  - it is only needed to import this class in the class of bean CallbackListener.
  - the signing of the method must be `public void method_name (Object)` where Object is the type of bean.

Lifecycle of the Beans 3.0 (2/3)

- Methods "callback" should be annotated with annotations:

```
@CallBackListener(String classname)
@PostConstruct
@PreDestroy
@PostActivate
@PrePassivate
@EntityListener(String classname)
@PrePersist
@PostPersist
@PreRemove
@PostRemove
@PreUpdate
@PostLoad
```

- Example 1: Definition of methods "callback" directly into the bean.

```
@Stateful
public class MyBean {
    private float total;
    private Vector productCodes;
    public int someMethods(){...};
    ...
    @PreDestroy aCallbackMethod() {...};
    ...
}
```

Lifecycle of the Beans 3.0 (3/3)

- Example 2: Definition of methods "callback" outside the bean.

First, we defines a class Listener containing methods "callback".

```
public class MyBeanListener {
    @PreDestroy aCallbackMethod() {...};
    ...
}
```

Then, we call the "callback listener" in the bean.

```
@CallbackListener MyBeanListener
@Stateful
public class MyBean {
    private float total;
    private Vector productCodes;
    public int someMethods(){...};
    ...
}
```

Injecting dependence (1/2)

- It allows a bean to acquire references to other resources or other objects of the application.
- Previously, he had to have recourse to XML files (with `ejb-ref`) and use the JNDI to access a resource:
  - To access to an EJB, it was mandatory to go through its interface Home to recover the use interface and its methods trades.
- Now is the container which injects these references before any method related to the life cycle of the EJB or business method is called.
- The container takes care to initialize all the resources the bean needs.

Injecting dependence (2/2)

- Example recovery of a resource-type database:

```
// It gets a resource myDB and it affects clientDB
// myDB type is deducted from the variable clientDB
@Ressource (name="myDB")
public DataSource clientDB;
```

- Example recovery of the reference of another EJB:

```
@EJB
public AddressHome addressHome;
```

- We can also go through the methods set ... to recover a resource:

```
// / / get the resource named clientDB name and of type DataSources
@Ressource(name="clientDB")
public void setDataSources(DataSources clientDB){
  this.sdd = clientDB;
}

// get the resource named myDB and of type DataSources
@Ressource
public void setClientBD(DataSources myDB){
  this.clientDB = myDB;
}
```

The interceptor methods

- These methods called "interceptors" will be performed at each call a method of defining these business methods.
- They are like:

```
public Object <methodName>(javax.ejb.InvocationContext)
    throws Exception
```

- Any interceptor method must take the following argument interface:

```
public interface InvocationContext {
  // returns the target bean of the interceptor method
  public Object getTarget ();
  // returns method triggering the interceptor
  public Method getMethod ();
  // return parameters of the business method
  public Object[] getParameters();
  public void setParameters (Object[] params);
  public java.util.Map<String, Object> getContextData();
  // Execute the following method interceptor or
  // method annotated @AroundInvoke
  public Object proceed() throws Exception;
}
```

Example of interceptor methods

```
@Stateless
@Interceptor("TestInterceptor")
public class LoginBean {
  @AroundInvoke
  public Object testInterceptor (InvocationContext invContext) throw Exception {
    invContext.proceed ();
  }
}

public class TestInterceptor {
  @AroundInvoke
  public Object myInterceptor (InvocationContext invContext) throws Exception{
    invContext.proceed ();
  }
}
```

Session Beans without state

- The class of Session Beans without state must be annotated with `@Stateless (import javax.ejb.Stateless;)`.
- These beans bear the callback events: `PostConstruct`, `PreDestroy`, but also `PostActivate` and `PrePassivate` which are like the methods `ejbActivate` and `ejbPassivate` in EJB 2.x.
- The callback method PostConstruct is working after all dependencies injections made by the container and before the first call of a business method.
- The PreDestroy callback method is called when the instance of the bean is destroyed.

Session Beans without state (Example) (1/2)

The remote interface of Bean:

```
package calc;

import javax.ejb.Remote;

@Remote public interface Calc {
    public double add(double val1, double val2);
    public double mult(double val1, double val2);
}
```

The class of the Bean:

```
package calc;

import javax.ejb.Stateless;

@Stateless public class CalcBean implements Calc {

    public double add(double val1, double val2) {
        return val1 + val2;
    }
    public double mult(double val1, double val2) {
        return val1 * val2;
    }
}
```

Session Beans without state (Example) (2/2)

The client of the bean:

```
import javax.naming.Context;
import javax.naming.InitialContext;
import calc.*;
import java.util.*;

public class CalcClient{
    public static void main(String[] args) throws Exception {
        Properties props = System.getProperties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
                  "org.jnp.interfaces.NamingContextFactory");
        props.put(Context.URL_PKG_PREFIXES,
                  "org.jboss.naming:org.jnp.interfaces");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        Context ctx = new InitialContext(props);
        Calc calc = (Calc) ctx.lookup("CalcBean/remote");

        double somme = 0;
        somme = calc.add(5.643, 8.2921);
        System.out.println("Addition de 5.643 + 8.2921 = "+somme);
    }
}
```

Session Beans with state

- The class must be annotated with `@Statefull (import javax.ejb.Statefull;)`
- These beans bear the callback events: `PostConstruct`, `PreDestroy`, `PostActivate` and `PrePassivate`.
- There is an annotation `@Remove` applicable to a business method, which involves the destruction of the bean after its call.

Session Beans with state (Example) (1/3)

The remote interface of the Bean:

```
package panier;

import javax.ejb.Remote;
import java.util.*;

@Remote public interface Panier{
    public void ajouterArticle(int idArticle);
    public void supprimerArticle(int idArticle);
    public Vector listerArticles();
    public void setNom(String nomClient);
    public String getNom();
    public void remove();
}
```

Session Beans with state (Example) (2/3)

The class of Bean:

```
package panier;

import javax.ejb.Stateful;
import javax.annotation.PostConstruct;
import javax.ejb.Remove;
import java.util.*;

@Stateful public class PanierBean implements Panier{
    Vector articles;
    String nomClient;

    @PostConstruct public void initialise() {
        articles = new Vector();
        nomClient = "";
    }

    public void ajouterArticle(int idArticle) {
        System.out.println ("Ajout d'un nouvel article");
        articles.add(new Integer(idArticle));
    }

    public void supprimerArticle(int idArticle){
        System.out.println ("Suppression d'un article");
        articles.remove(new Integer(idArticle));
    }

    public Vector listerArticles(){
        return articles;
    }

    public void setNom(String nomClient) {
        this.nomClient = nomClient;
    }

    public String getNom() {
        return nomClient;
    }

    @Remove public void remove() {
        articles = null;
```

Session Beans with state (Example) (3/3)

The client of the bean:

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;
import panier.*;

public class PanierClient {
    public static void main(String[] args) throws Exception {
        Properties props = System.getProperties();
        props.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        props.put(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        Context ctx = new InitialContext(props);
        Panier monPanier = (Panier) ctx.lookup("PanierBean/remote");
        monPanier.ajouterArticle(65);
        monPanier.ajouterArticle(53);

        Vector mesArticles = monPanier.listerArticles();
        System.out.println ("Il y a "+mesArticles.size()+" article(s) dans le panier !");
        Enumeration e = mesArticles.elements();
        while (e.hasMoreElements()) {
            System.out.println((Integer)e.nextElement());
        }

        monPanier.ajouterArticle(23);
        monPanier.ajouterArticle(18);
        monPanier.supprimerArticle(65);

        mesArticles = monPanier.listerArticles();
        System.out.println ("Il y a "+mesArticles.size()+" article(s) dans le panier !");
        e = mesArticles.elements();
        while (e.hasMoreElements()) {
            System.out.println((Integer)e.nextElement());
        }
        monPanier.remove ();
    }
}
```

Message Driven Beans

- Class of bean should be annotated with `@MessageDriven` (`import javax.ejb.MessageDriven;`).
- The interface of the bean must be of the type that will use the bean. In most cases, it will be `javac.jms.MessageListener`.
- These beans bear the callback events: `PostConstruct` and `PreDestroy`.
- In the case of use with JMS, the method `public void onMessage (Message msg)` must be redefined.
- We must also define the parameters to be used for connecting to the destination.
- For this, we use the properties: destinationType and destination.

Message Driven Beans: example (1/2)

- The class of the Bean:

```
import javax.annotation.Resource;
import javax.ejb.*;
import javax.jms.*;

@MessageDriven(activationConfig =
{
  @ActivationConfigProperty(propertyName = "destinationType",
                            propertyValue = "javax.jms.Queue"),
  @ActivationConfigProperty(propertyName = "destination",
                            propertyValue = "queue/MessageBean")
})
public class MessageBean implements MessageListener {
  @Resource MessageDrivenContext mdc;
  //Méthode de réception des messages
  public void onMessage(Message msg) {
    TextMessage tm;
    try {
      if (msg instanceof TextMessage){
        tm = (TextMessage) msg;
        String text = tm.getText(); System.out.println("Message Bean Reçu: " + text);
      }
      else {
        System.out.println ("Message de mauvais type: "+msg.getClass().getName());
      }
    }catch(JMSException e) {
      e.printStackTrace();
      mdc.setRollbackOnly ();
    }catch(Throwable te) {
      te.printStackTrace();
    }
  }
}
```

Message Driven Beans: example (1/2)

- The client:

```
import javax.naming.*;
import javax.jms.*;
import java.util.*;

public class Client {

  public static void main (String[] args) throws Exception {
    // Initialisation JNDI
    Properties props = System.getProperties();
    props.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
    props.put(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");
    props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

    Context ctx = new InitialContext(props);
    // 1: recherche d'une fabrique de connection via  JNDI
    QueueConnectionFactory factory = (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
    // 2: Utilisation de la fabrique de connexions pour créer une connexion JMS
    QueueConnection connection = factory.createQueueConnection();
    // 3: Utilisation de la connection pour créer une session
    QueueSession session = connection.createQueueSession(false, QueueSession.AUTO_ACKNOWLEDGE);
    // 4: Recherche du sujet (topic) via JNDI
    javax.jms.Queue queue = (javax.jms.Queue) ctx.lookup("queue/MessageBean");
    // 5: Création d'un producteur de message
    QueueSender sender = session.createSender(queue);
    // 6: Creation d'un  message texte et publication
    TextMessage msg = session.createTextMessage();
    for (int i=0;i<10;i++){
      msg.setText("Envoi du message: " + i);
      sender.send(msg);
    }
  }
}
```

Entity Beans (1/5)

- They also POJOs but their specifications have been defined in part because they have undergone many changes in the standard 3.0.
- The annotation `@Entity (import javax.persistance.Entity;)` defines the bean as an Entity Bean.
- The bean must have at least one constructor default and will inherit the interface Serializable to be used throughout the network for the management of persistence.
- You can specify two different methods for managing the persistence with the option *access*:
  - `@Entity(access=AccessType.FIELD)` allows access directly to the fields to make persistent.
  - `@Entity(access=AccessType.PROPERTY)` requires the supplier to use the accessors..

Entity Beans (2/5)

- The primary key can be simple or compound and must be declared with the annotation `@Id`.
  - For example, to obtain a key that automatically increases: `@Id(generate=GenerateType.AUTO)`.
- For the composed keys, we must respect certain principles:
  - The class of the primary key must be public and have a constructor without arguments.
  - If access is type PROPERTY, the class can be either type public is protected type.
  - The class must be serializable (implement Serializable).
  - Implementation methods equals () and hashCode ().
  - If the primary key is mapped to multiple fields or properties, the names of key fields of this must be the same ones used in the Entity Bean.
- The annotations can make the object / relational mapping and management of relations between the entities.

Entity Beans (3/5)

- When creating an Entity Bean, you must make the mapping of all its fields. A mapping default occurs when annotation above the field: `@Basic` specifies this behavior.
- `@Table` sets the table for the class, she takes as an argument the name of the table.

```
@Entity(access=accessType.FIELD)
@Table(name="PAYS")
public class Pays implements Serializable {
  @Id(generate=GeneratorType.AUTO) private int id;
  @Basic private String nom;

  public Pays() {
  }

  public Pays(String nom) {
    this.nom = nom;
  }

  public int getId() {
    return id;
  }
}
```

Entity Beans (4/5)

- It can be matched to a corresponding value to a specific field of the database using the annotation `@Column` and options as `name`, which specifies the name of the column, or options to determine if field can be null,...

```
@Column(name="DESC", nullable=false)
public String getDescription() {
  return description;
}
```

- It exists some relations `OneToOne`, `OneToMany`, `ManyToOne`, `ManyToMany` (defined by annotations). In these cases, we must not forget to specify the columns forming the joints.

```
@ManyToOne(optional=false)
@JointColumn(name = "CLIENT_ID", nullable = false, updatable = false)
public Client getClient (){
  return client;
}
```

Entity Beans (5/5)

- The Entity Beans manipulated through an `EntityManager`.
- This `EntityManager` can be obtained from a Session Bean by injection of dependency.

```
@Stateless public class EmployeeManager{
  @Resource EntityManager em;

  public void void updateEmployeeAddress (int employeeId, Address address) {
    //Recherche d'un bean
    Employee emp = (Employee)em.find ("Employee", employeId);
    emp.setAddress (address);
  }
}
```

Entity Beans: example (1/12)

- Main principle:
  - It creates an Entity Bean that will handle persistent objects.
  - In the bean entity, it establishes a mapping between the attributes of the bean and a table of the database.
  - The client does not access directly to the Entity Beans but beans goes through sessions which will perform the necessary manipulations on Entity Beans.
- The bean of the example:
  - `ArticleBean`: it is the Entity Bean.
  - `ArticleAccessBean`: it is a Session Bean without state. It is composed of the class `ArticleAccessBean.java` and teh interface `ArticleAccess.java`.

Entity Beans: example (2/12)

- The interface `ArticleAccess.java`: definition of the business methods.

```
package article;

import javax.ejb.Remote;
import java.util.*;

@Remote public interface ArticleAccess {
  public int addArticle (String libelle, double prixUnitaire, String categorie);
  public void delArticle (int idArticle);
  public InfosArticle rechercherArticle (int idArticle);
  public List rechercherTheArticlesParCategorie (String categorie);
  public List rechercherTousTheArticles ();
}
```

Entity Beans: example (3/12)

- The class `ArticleAccessBean.java`: the method `addArticle`

```
package article;

import javax.ejb.*;
import javax.persistence.*;
import java.util.*;

@Stateless public class ArticleAccessBean implements ArticleAccess {

  @PersistenceContext(unitName="Articles")
  EntityManager em;

  public int addArticle (String libelle, double prixUnitaire, String categorie){
    ArticleBean ab = new ArticleBean ();
    ab.setCategorie (categorie);
    ab.setLibelle (libelle);
    ab.setPrixUnitaire (prixUnitaire);
    em.persist(ab);
    em.flush();
    System.out.println ("AccessBean: Ajout de l'article "+ab.getIdArticle ());
    return ab.getIdArticle ();
  }
```

Entity Beans: example (4/12)

- The class `ArticleAccessBean.java`: the method `delArticle`

```
public void delArticle (int idArticle){
      Query query = em.createQuery("DELETE FROM ArticleBean AS a WHERE a.idArticle="+idArticle);
}

public InfosArticle rechercherArticle (int idArticle){
  Query query = em.createQuery("SELECT a FROM ArticleBean AS a WHERE a.idArticle="+idArticle);

  List<ArticleBean> allArticles = query.getResultList();

  ArticleBean article = allArticles.get(0);
  return new InfosArticle (article.getIdArticle(), article.getLibelle(),
                           article.getPrixUnitaire(), article.getCategorie());
}
```

Entity Beans: example (5/12)

- The class `ArticleAccessBean.java`: the method `rechercherTousTheArticles`

```
public List rechercherTousTheArticles (){
      Query query = em.createQuery("SELECT a FROM ArticleBean AS a");
      List<ArticleBean> articlesbean = query.getResultList();
      Vector<InfosArticle> articles = new Vector();
      Iterator i = articlesbean.iterator();
      ArticleBean article;
      while (i.hasNext()) {
          article = (ArticleBean) i.next();
          InfosArticle infos = new InfosArticle (article.getIdArticle(), article.getLibelle(),
                                                article.getPrixUnitaire(), article.getCategorie());
          articles.add(infos);
      }
      return articles;
 }
```

Entity Beans: example (6/12)

- The class `ArticleAccessBean.java`: the method `rechercherTheArticlesParCategorie`

```
  public List rechercherTheArticlesParCategorie (String categorie){
        Query query = em.createQuery("SELECT a FROM ArticleBean AS a WHERE categorie='"+categorie+"'");
        List<ArticleBean> articlesbean = query.getResultList();
        Vector<InfosArticle> articles = new Vector();
        Iterator i = articlesbean.iterator();
        ArticleBean article;
        while (i.hasNext()) {
            article = (ArticleBean) i.next();
            articles.add(new InfosArticle (article.getIdArticle(), article.getLibelle(),
                                           article.getPrixUnitaire(), article.getCategorie()));
        }
        return articles;
   }
}
```

Entity Beans: example (7/12)

- The file of the persistence management: `persistence.xml`

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                       http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
   version="1.0">
   <persistence-unit name="Articles">
      <jta-data-source>java:/DefaultDS</jta-data-source>
      <properties>
         <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
         <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      </properties>
   </persistence-unit>
</persistence>
```

Entity Beans: example (8/12)

- The client `ArticleClient.java`: the method `creerArticle`

```
import javax.naming.*;
import javax.rmi.*;
import javax.ejb.*;
import java.util.*;
import article.*;

public class ArticleClient {

  public void creerArticle(String libelle, double montantUnitaire, String categorie) {
        ArticleAccess ah;
        Properties props = System.getProperties();
        props.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
        props.put(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        try {
            Context ctx = new InitialContext(props);
            ah = (ArticleAccess) ctx.lookup("ArticleAccessBean/remote");
            System.out.println ("Ajout d'un article: "+ah);
            int id = ah.addArticle(libelle, montantUnitaire, categorie);
            System.out.println ("Affichage de l'article "+id);
            afficherArticle(id);
        } catch (Throwable th) {
            System.out.println("Erreur dans creerArticle: " + th);
        }
   }
```

Entity Beans: example (9/12)

- The client `ArticleClient.java`: the method `afficherArticle`

```
    public void afficherArticle(int numeroArticle) {
        ArticleAccess ah;
        Properties props = new Properties();
        props.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
        props.put("java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces");
        props.put("java.naming.provider.url", "localhost:1099");
        try {
            Context ic = new InitialContext(props);
            ah = (ArticleAccess) ic.lookup("ArticleAccessBean/remote");
            InfosArticle infos = ah.rechercherArticle(numeroArticle);
            System.out.println("voici les infos sur l'article: " + infos.idArticle);
            System.out.println("   id: " + infos.idArticle);
            System.out.println("   libelle: " + infos.libelle);
            System.out.println("   prix unitaire: " + infos.prixUnitaire);
            System.out.println("   categorie: " + infos.categorie);
        } catch (Throwable th) {
            System.out.println("GereCommande.creerArticle: " + th);
        }
    }
```

Entity Beans: example (10/12)

- The client `ArticleClient.java`: the method `afficherArticlesParCategorie`

```
   public void afficherArticlesParCategorie(String categorie) {

        ArticleAccess ah;
        Properties props = new Properties();
        props.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
        props.put("java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces");
        props.put("java.naming.provider.url", "localhost:1099");
        try {
            Context ic = new InitialContext(props);
            ah  = (ArticleAccess) ic.lookup("ArticleAccessBean/remote");
            List<InfosArticle> articles = ah.rechercherTheArticlesParCategorie(categorie);
            Iterator i = articles.iterator();
            InfosArticle article;
            while (i.hasNext()) {
                article = (InfosArticle) i.next();
                afficherArticle(article.idArticle);
            }
        } catch (Throwable th) {
            System.out.println("Erreur dans rechercherArticlesParCategorie: " + th);
        }
    }
```

Entity Beans: example (11/12)

- The client `ArticleClient.java`: the method `afficherTousTheArticles`

```
 public void afficherTousTheArticles() {

      ArticleAccess ah;
      Properties props = new Properties();
      props.put("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
      props.put("java.naming.factory.url.pkgs", "org.jboss.naming:org.jnp.interfaces");
      props.put("java.naming.provider.url", "localhost:1099");
      try {
          Context ic = new InitialContext(props);
          ah  = (ArticleAccess) ic.lookup("ArticleAccessBean/remote");
          List<InfosArticle> articles = ah.rechercherTousTheArticles();
          Iterator i = articles.iterator();
          InfosArticle article;
          while (i.hasNext()) {
              article = (InfosArticle) i.next();
              afficherArticle(article.idArticle);
          }
      } catch (Throwable th) {
          System.out.println("Erreur dans rechercherArticlesParCategorie: " + th);
      }
  }
```

Entity Beans: example (12/12)

- The client `ArticleClient.java`: the method `main`

```
    public static void main(java.lang.String[] args) {

        ArticleClient a = new ArticleClient ();

        a.creerArticle("The miserables", 21, "LIVRE");
        a.creerArticle("Celine Dion au stade de France", 120, "CD");
        a.creerArticle("Je l'aime a mourir", 28, "LIVRE");
        a.creerArticle("La mer", 38, "LIVRE");

        // Recherche de l'article 3
        System.out.println ("================================");
        a.afficherArticle(3);
        System.out.println ("================================");
        a.afficherTousTheArticles();
        System.out.println ("================================");

        // Recherche de la categorie
        a.afficherArticlesParCategorie("CD");
        System.out.println ("================================");

    }
}
```