Enterprise Development Lecture 5

Enterprise Java Beans (EJB's)

Objective

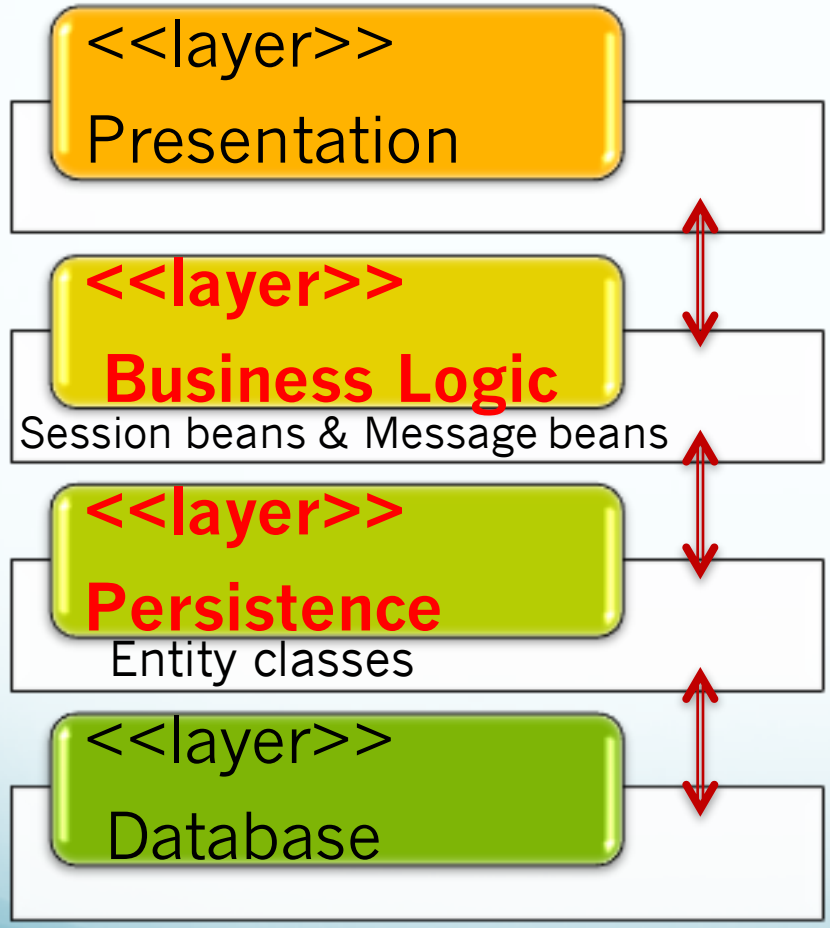
How are distributed applications enabled and supported by Java EE.

- EJB's
- Local and remote interfaces
- EJB containers
  - Services offered by EJB container
- Session and Message beans

EJB's – Enterprise Java Beans

Coding the middleware / business logic tier

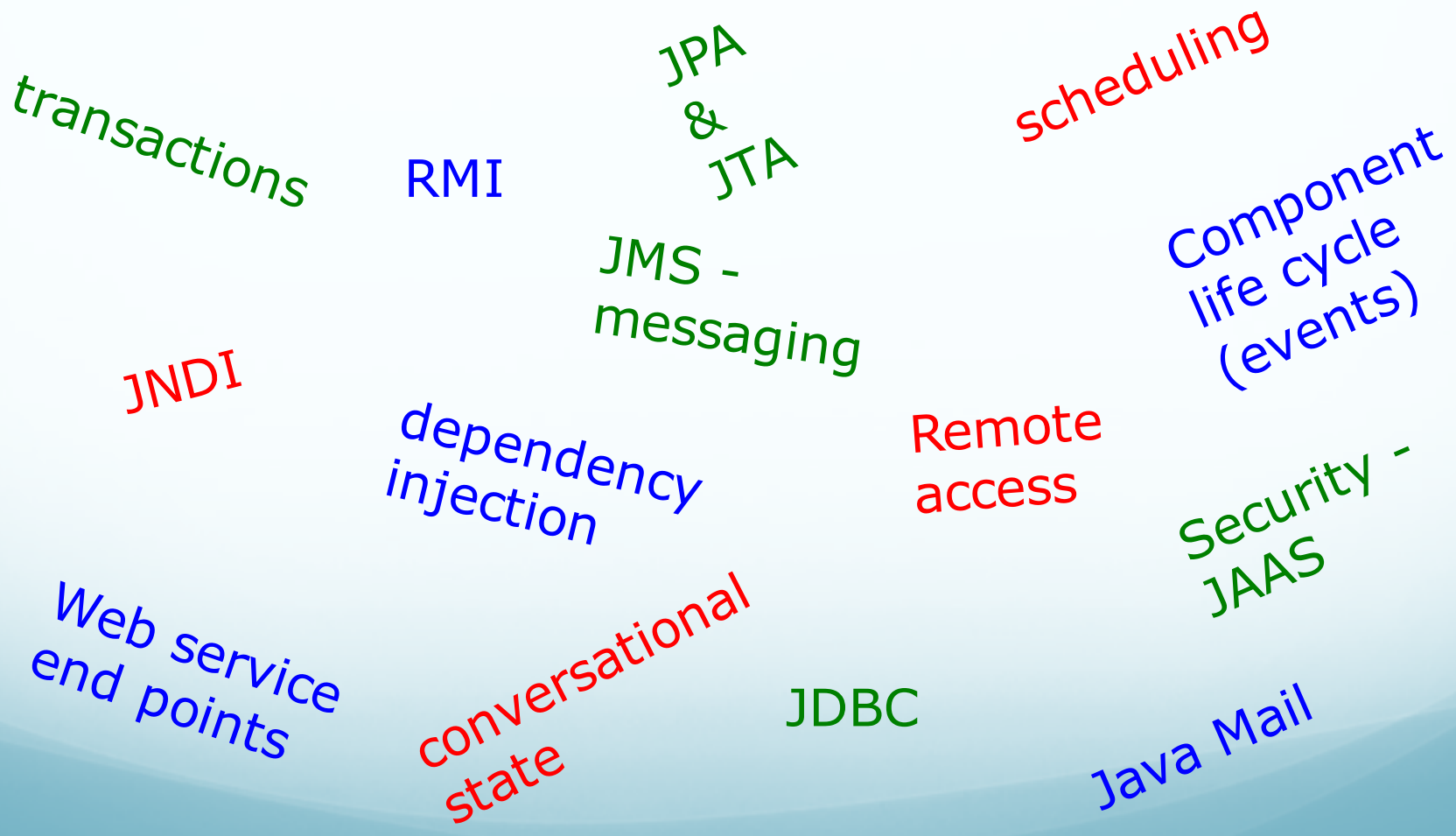
EJBs



- EJB's can handle a greater range of business requirements than POJO's (plain old java objects)

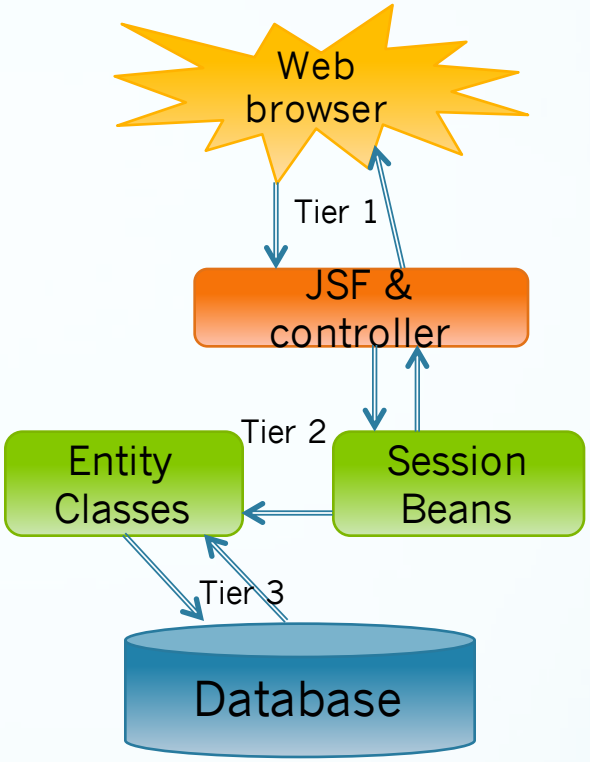
EJB's

- Just as JSF / PHP pages run in a web server, EJB's run in an EJB container, which supports:



Types of EJB's

- There are two types of EJB's:
  - **Session beans**: general purpose, transaction aware, non persistent beans used to model processes, services, and client-server sessions
  - **Message beans**: listeners for asynchronous messages delivered via the JMS – java messaging service. They are often used to handle interactions with a legacy system or business-to-business communication. They model workflows.



Session Beans

- Session beans are where the programming is done. These implement the business logic including:
  - Adding data to the table by creating an instance of the corresponding entity bean
  - Updating data by calling the corresponding set methods
  - Reading data using the get methods
  - Querying the database, for example to see if there is enough stock to fill an order
- . . .and any other functionality required by the application.

Session beans

A session bean may be stateful, stateless or singleton.
The state of an object refers to the **values of its instance variables**

Stateful

- The session bean instance has one client only.
- Maintains conversation state for the duration of a session
  - the variables keep their values for the duration of a session

Stateless

- The session bean can be shared by many clients.
- It does NOT maintain conversation state with a client.
  - Variables keep their values for the duration of the method call only.

Singleton

Can only have one instances shared by all clients.
State is maintained across client invocations.

Stateful session beans

- An instance of a stateful session bean is dedicated to the Java client that created it.
- It's class defines instance variables that can hold session data from one method invocation to the next, which makes methods inter-dependent.
- Stateful session beans are used to model business concepts that represent agents or roles that perform tasks on a client's behalf.
  - Examples of such beans may include a bank teller that processes monetary transactions, a shopping cart used in on-line commerce, or a travel agent that makes flight, hotel, and car-rental reservations.

Example of a stateful session bean:

- A program implementing methods for a shopping cart with include methods such as:
  - addProduct()
  - removeProduct()
  - updateQty()
  - completePurchase()
- If the cart is stored in memory, actions would be specific to a particular customer, and so an instance can not be shared by other clients.

```
@Stateful
@StatefulTimeout(unit = TimeUnit.MINUTES, value =
30)

public class ShoppingCartBean {

@PersistenceContext
private EntityManager em;

private List<Product> products;

private void init(){
        products = new ArrayList<>(); }

public void addProduct(Product product){
        products.add(product); }

public void completePurchase(){
   for(Product product : products){
        em.persist(product); }
   products.clear(); }
}
```

Stateless Session beans

- Stateless session beans are not dedicated to a single client; they are shared by many.
- This makes stateless session beans very **lightweight** and fast, but also limits their behavior.
- Stateless session beans behave more like an API, where method invocations are **not** interdependent. Each invocation is effectively stateless and is analogous to static methods in a Java class.
  - Stateless session beans are used to model stateless services, for example: credit-card processing, financial calculations, library searches.

Example of a stateless session bean

- A program implementing a credit card service which charges a particular amount to a specified credit card provides a ONCE off service based on data passed in as an argument.
- It's instance can be shared by many clients.
- It does NOT need to maintain conversation state with a client.

```
@Stateless

public class CreditCardBean {

@PersistenceContext
private EntityManager em;


public void addPurchase(Purchase
purchase){
        em.persist(purchase); }

 . . .

}
```

Singleton session bean

- As the name suggests, a Singleton is a session bean with a guarantee that there is at most one instance in the application, in accordance with the singleton design pattern.
- A singleton session bean is instantiated once per application, and exists for the lifecycle of the application.
- Uses:
  - It provides a place to hold data that pertains to the entire application and all users using it.
  - to perform tasks upon application startup and shutdown

Single session bean example:

Count how many times a web page is requested:

- The JSF / Controller bean calls getHits() which increments hits.
- The value of hits persists for the life time of the application.

```
package singleton.counter;
import javax.ejb.Singleton;
/** * CounterBean is a simple singleton session bean that records the number * of hits to a web page. */
@Singleton

public class CounterBean {

private int hits = 1;

    // Increment and return the number of hits
    public int getHits() {
       return hits++;
    }
}
```

Implementing a session bean

A session bean is a standard Java class with the following requirements:

1. The class must be annotated with @Stateless, @Stateful, or @Singleton
2. The class must be defined as public (cannot be final or abstract)
3. The class must have a public, no arg constructor that the container will use to create instances
4. The class must not define the finalize() method

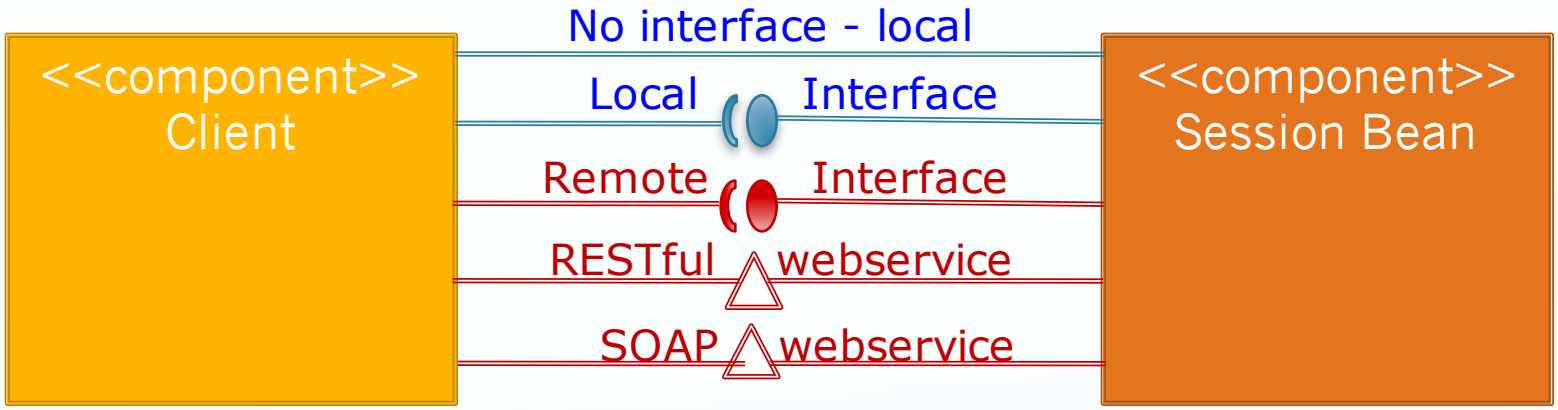
5. It is recommended that client access is via an interface rather than directly to the bean.

   This **loose coupling** makes it easier to modify the bean without effecting client code.

Access to a session bean

- A session bean can be called in three ways:
  1. Via the **no-interface view**: created WITHOUT an interface for access from clients in the same application.
     - Can access public methods only
  2. Via one of two business interfaces
     - **Local interface** (more efficient than no-interface for clients in the same application)
     - **Remote interface** (client can be local or remote. May be slower than local interface.)
  3. As a **web service,** remote access for both Java and non Java based clients – technology independent.

EJB Interfaces



Local interface is generally used between session beans in the same applications that work together to implement a process or workflow.

Remote interface or web service is used by a client application.

EJB Interfaces

The interfaces can be specified in the session bean class using annotations:

Local and remote interfaces

```
@Local
Public interface ItemLocal {
    List<Book> findBooks();
    List<CD> findCDs();
}

@Remote
Public interface ItemRemote {
    List<Book> findBooks();
    List<CD> findCDs();
}
```

Stateless session bean implementing the local and remote interfaces

```
@Stateless
@Remote (ItemRemote)
@Local (ItemLocal)
@WebService

public class ITEMEJB implements
     ItemLocal, ItemRemote {

public List<Books> findBooks(){

     //code to call findAll books }

public List<CD> findCDs(){

     //code to call findAll cds }

}
```

Client view

- The client of a session bean can be any kind of POJO, graphical interface (SWING), a servlet/JSP, a JSF-managed bean, a web service or another EJB (deployed on the same or on a different container).
- To invoke a session bean, the client references the bean, or it's interfaces, using JNDI or using the @EJB annotation.
  - @EJB injects session bean references into the client code. This dependency injection will only work within managed environments, such as web containers (e.g. Tomcat), EJB containers or application containers (e.g. glassfish).

Client view

Session Bean

```
@Stateless

@Local (ItemLocal)

. . . . .

public class ItemEJB
    implements ItemLocal {

 . . .  .

}
```

Client Code

Using annotation:

```
@EJB ItemLocal itemEJBLocal;
```

Using JNDI:

```
java.lang.Object ejbLocal =
    initialContext.lookup(java:global/le
    ct5app/ItemEJB!ItemLocal
```

Note: JNDI string is explained on slide 24

Terminology . . .

- A session bean can also be referred to as a session façade
- Session façade is one design pattern that is often used while developing enterprise applications.
- It is implemented as a higher level component (i.e.: Session Bean), and it contains all the interactions between low level components (i.e.: Entity class). It then provides a single interface for the functionality of an application or part of it, and it decouples lower level components simplifying the design.

Some of the services available to a session bean . . .

1. Accessing a resource

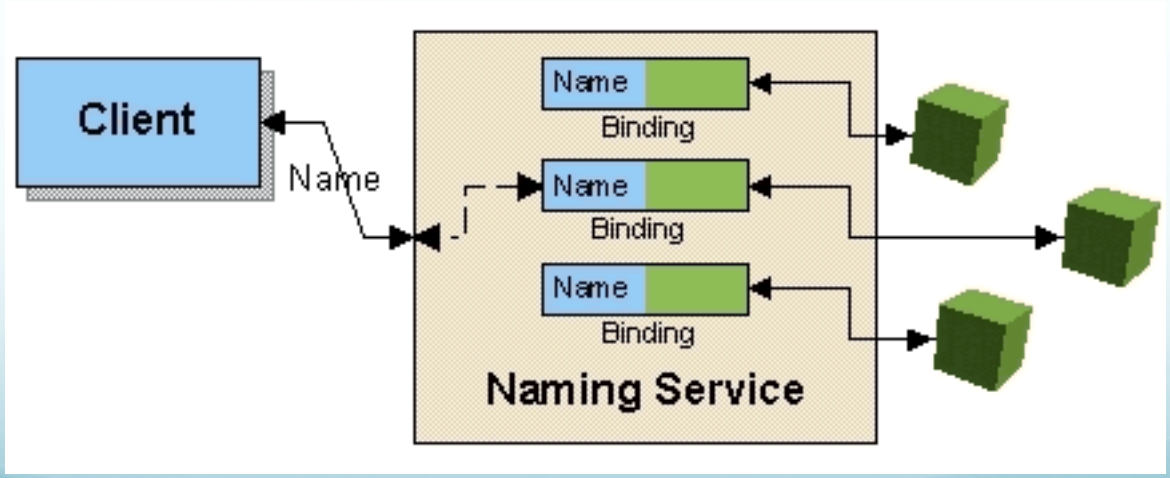
- Resources available to a bean include other EJB's, web services, datasources, environment resources, mail services, timer services etc.
- Interaction between resources should be loosely coupled. This is typically done by implementing interfaces for each resource.
  - A bean will interact with an interface rather than the objects themselves, allowing implementations to change without effecting the interface definition, or calls to it.

1. Accessing a resource

- Access to a resource can be implemented in one of two ways:
  - Using Java Naming and Directory Service (JNDI) directly
    - Object Pulls a reference to the resource from the container at run time
    - More difficult to code
  - Using Dependency Injection – DI (also referred to as IoC – Inversion of Control).
    - Container Pushes a reference for the resource onto the object at deployment time (whether its used or not)
    - Simpler code
    - Only available in beans managed by a container.

JNDI

- JNDI, Java Naming and Directory Service, is an application programming interface (API) for accessing different kinds of naming and directory services, such as LDAP.
- A naming and directory service maintains a list of available resources (EJB's, web services, datasources, JMS destinations, environment resources, etc.)



JNDI

- Each resource object is identified by a unique, people-friendly name, called the JNDI name.
- A resource object and its JNDI name are bound together by the naming and directory service, which is included with the Application Server.
- JNDI names are organised in a hierarchical structure. How you reference a resource depends on where you are calling from:
  - Within the same module: java:module/<bean-name>[!interface-name]
  - Within the same application: java:app/<module-name>/<bean-name>[!interface-name]
  - From outside the application: *java:global[/app-name>]/<module-name>/<bean-name>*[!interface-name]

Note: Modules are discussed again on slide 40. All code in a particular module of an application runs in the same container

JNDI

- JNDI services are accessed through the initialContext class, which provides methods such as:
  1. void bind(String stringName, Object object): Bind a name to an object.
  2. Object lookup(String stringName): Return a specified object.
  3. void unbind(String stringName): Unbinds the specified object

- JEE components locate objects by invoking the JNDI lookup method, for example:

```
Context ctx = new InitialContext();
BookEJB bookEJB = (BookEJB)
   ctx.lookup("java:global/chapter06/BookEJB");
```

Dependency Injection

- Dependency injection simplifies access to resources by using annotations instead of direct calls to JNDI
- For example:

```
@Resource
private javax.sql.DataSource AdventureDB;

@EJB
Private static BookEJB bookEJB
```

- Access to the resource can be configured in an XML file.

2. Accessing a service

- The session bean can avail of services provided by the container via the **session context**.

```
@Resource
private SessionContext context;
```

- Most services are handled transparently by the container on behalf of the bean
  - e.g there is no need to explicitly mark a transaction to issue a commit/rollback

- Sometimes the bean needs to explicitly use container services. Methods include: lookup(String) – lookup a resource; isCallerInRole (Test if the caller has a given security role), setRollbackOnly (mark for rollback), getTimerService (access timer services ...)

More details:
http://docs.oracle.com/javaee/6/api/javax/ejb/EJBConte

The Timer Service

- The TimerService interface provides enterprise bean components with access to the container-provided Timer Service.
- This allows methods within a bean to be called at specific times, or specific time intervals.
- There is a rich syntax for expressing when and how often a method should be called – i.e. calendar expression.
- The @schedule annotation signifies a method is to be called by the timer service.

Examples

```
@stateless
Public class exampleEJB {

//generate report at 5:30am on the 1st day of each month
@schedule (dayOfMonth="1", hour="5", minute="30")
public void generateReport() { . . . . }

//do a backup each day at 23 hours, Paris time.
@schedule (dayOfMonth="*", hour="23",
    timezone="Europe/Paris")
public void callBackup()

//On the second Tuesday of each month, take a sample of the
    dataset every 2 hours, starting at 12 noon.
@schedule (hour="12/2", dayOfMonth="2nd Tue")
public void sampleDataset()
```

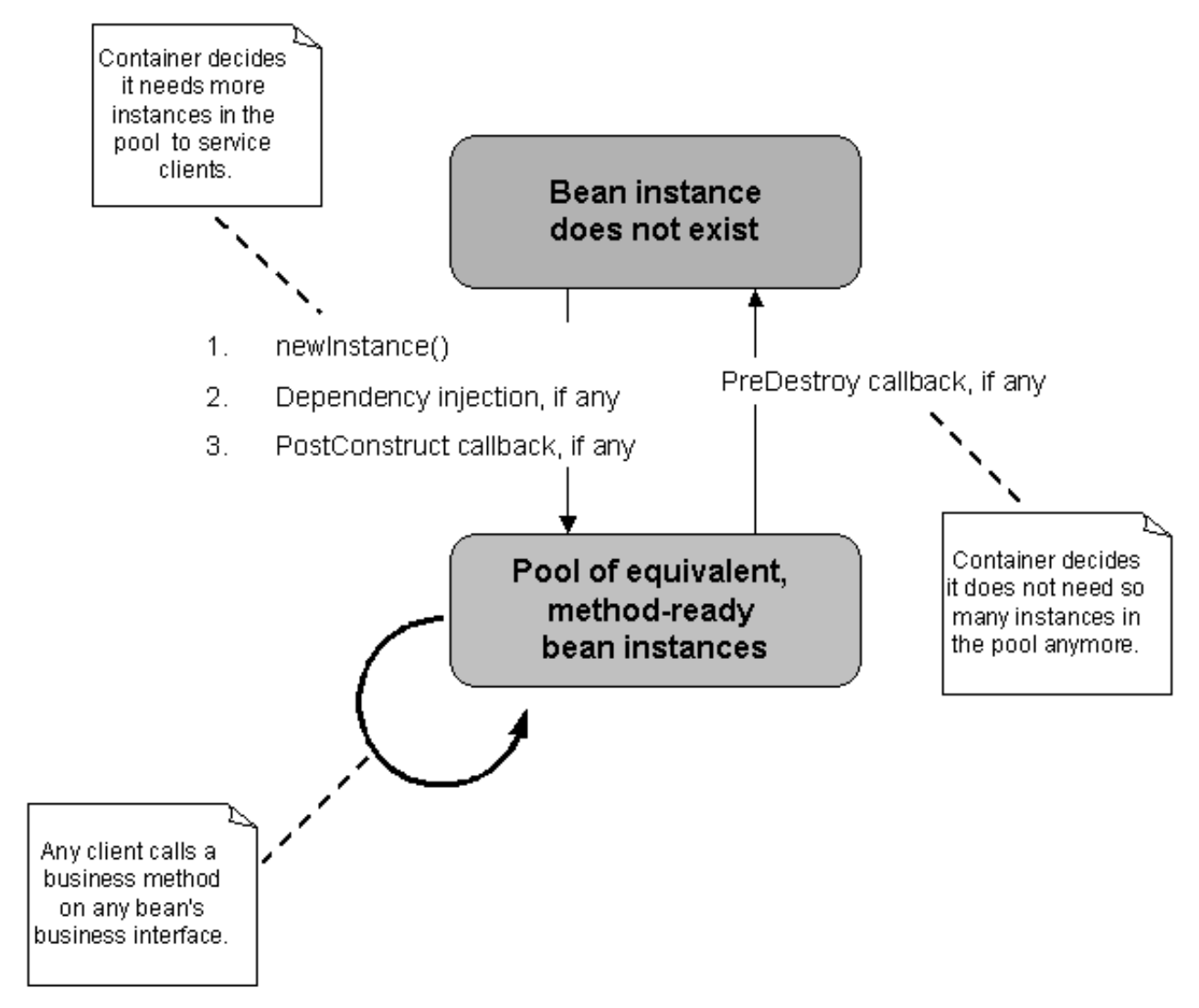
3. Resource Management

- An EJB container must be able to handle heavy workloads
  - Support many users using thousands of objects simultaneously
- There are two mechanisms that improve performance for heavy workloads
  1. Instance pooling
  2. Activation

Instance Pooling

- Instance pooling is managing a collection of bean instances so that they are quickly accessible at runtime.
- An EJB container creates serveral instances of the bean, in anticipation of client requests.
- As clients make business method requests, bean instances from the pool are assigned to the interface associated with the client.
- When the client doesn't need the instance any more, or time has elapsed since the last request, the instance is return to the pool.
- Instances are selected arbitrarily from the pool
- Instance pooling can be used for stateless and singleton session beans.

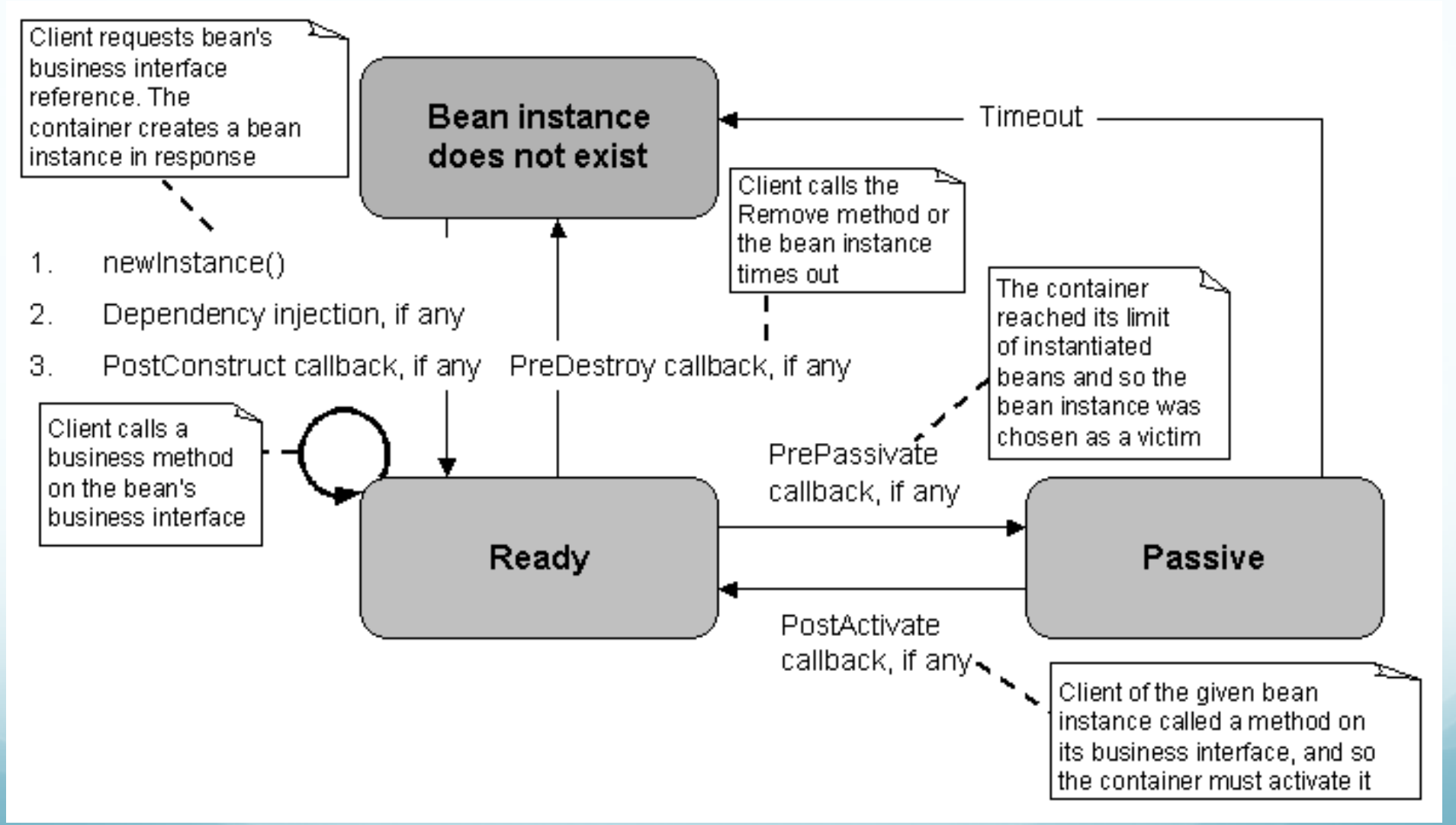
Stateless session bean life cycle



Activation

- Instances of Stateful session beans are created as a result of a client request, and are used exclusively by that client for their lifetime- maintaining conversational state.
- The integrity of this conversational state needs to be maintained for the life of the beans service to the client
- Therefore they can not be pooled like other beans
- To conserve resouces, stateful session beans can be **evicted** from memory, i.e. the container passivates the bean.
- To **passivate** the bean, the beans state is stored to secondary storage, and maintained relative to the client
- When the client invokes a method, a **new** instance is instantiated, and populated from the passivated secondary storage **– activating a bean**

Stateful session bean - life cycle



Message beans

Also called Message-Driven beans

Message bean

- Session beans are primarily used for synchronous communications.
- Sometimes, the communication between a client and server does not need an immediate response, in which case communication can be asynchronous, for example:
  - Sending an e-mail
  - Checking assessments for plagiarised content (turn-it-in)
- While Session Beans can send messages, they can not listen for, and receive asynchronous messages. That is the role of a message bean.

Message bean

- Message beans are similar to session beans in terms of the resources and services they can access, but with one important difference:
  - A session bean's methods can be called directly by the client (via an interface)
  - Methods is a message bean can not be called directly. A message bean listens for messages, and on receiving a message invokes the appropriate method.
- A JMS (Java Message Services) manages the queuing and routing of the messages themselves.
  - Session beans can send an asynchronous call/message to a JMS server for processing. Only message beans can receive messages from the message server.

Message bean

Message beans have the following characteristics:

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- They can access and update data in a database.
- They can be transaction-aware.
- They are stateless.

Message bean

- When a message arrives, the container calls the message-driven bean's **onMessage** method to process the message. The onMessage method normally casts the message to one of the five JMS message types and handles it in accordance with the application's business logic. The 5 types are:
  - Object (serialisable Java Object)
  - Byte (an array of bytes)
  - Text (a String)
  - Stream (sequence of primitive Java types)
  - Map (name value pairs)
- The **onMessage** method can call helper methods or can invoke a session bean to process the information in the message or to store it in a database.

Example of onMessage

to add a row to a database table storing news articles

```
public void onMessage(Message message) {
        ObjectMessage msg = null;


        if (message instanceof ObjectMessage) {
            msg = (ObjectMessage) message;
            NewsEntity e = (NewsEntity) msg.getObject();
            em.persist(e);
 }
}
```

Checking the message is a Java object, which this bean is expecting

Casting the message as an instance of the NewsEntity entity class

Calling a session bean method to add a new row to the NewsEntity database table

Sending a message to a message bean:

```
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageProducer messageProducer = session.createProducer(queue);

ObjectMessage message = session.createObjectMessage(); // here we create

NewsEntity, that will be sent in JMS message
NewsEntity e = new NewsEntity();
e.setTitle(title);
e.setBody(body);

message.setObject(e);
messageProducer.send(message);

messageProducer.close();
connection.close();
```

Link to a message queue

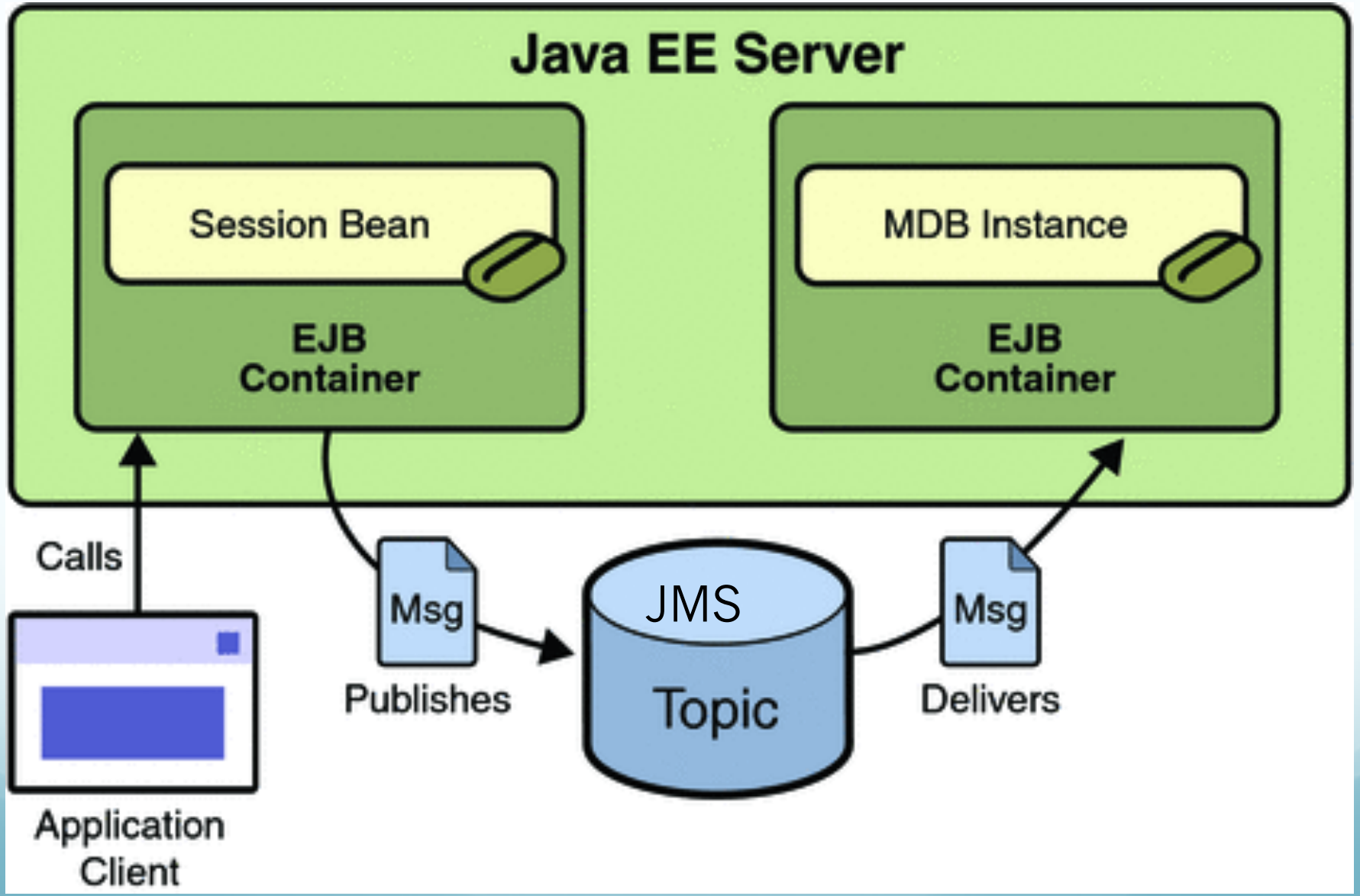
Create a message object

Create content of message – in instance of a NewsEntity object

Add the content to the message and send the message

Close the connect to the message queue

Message driven bean

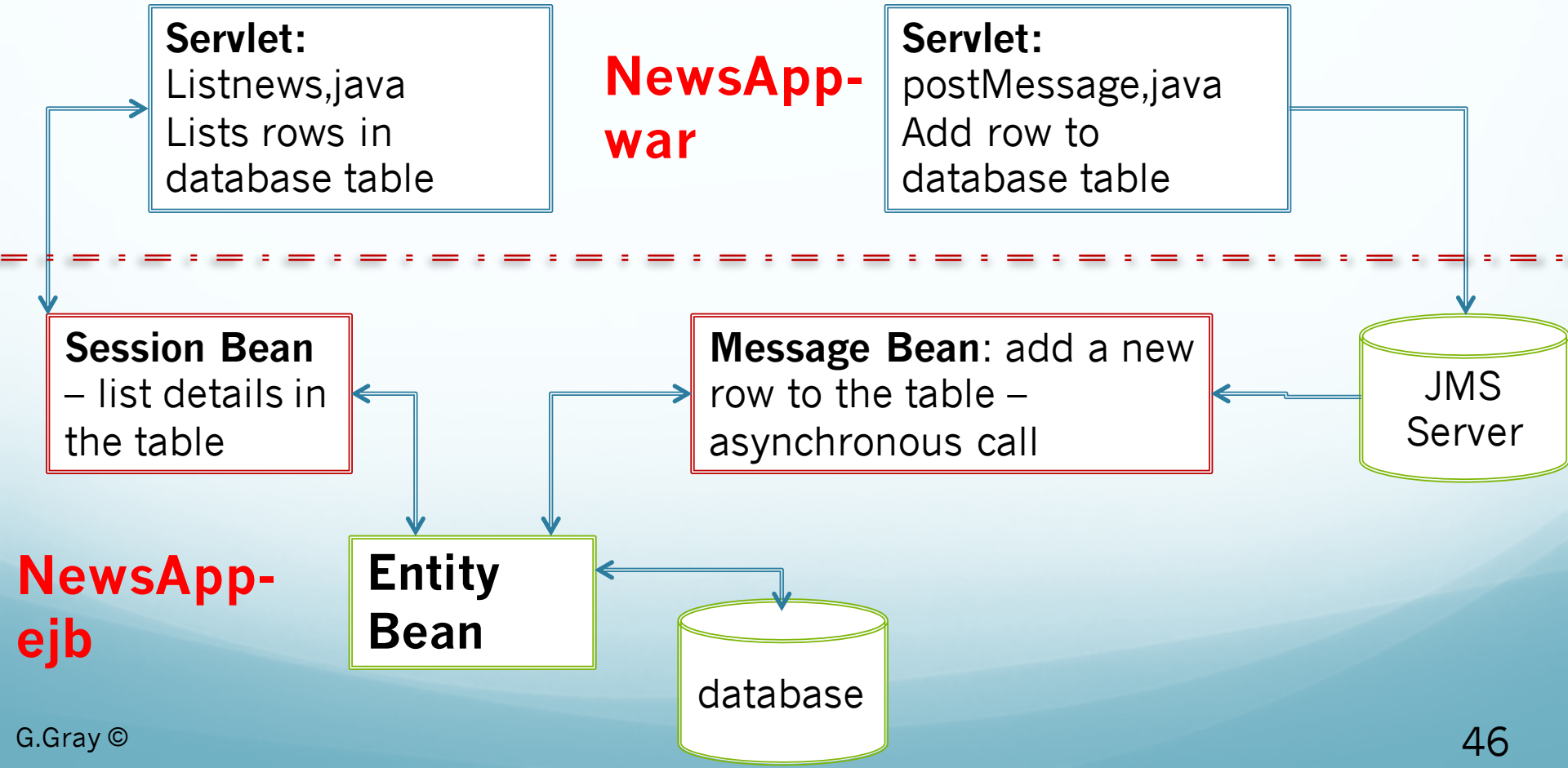


Building a JEE application in Netbeans

- Java EE applications are built as a collection of modules, each with its own project node in Netbeans. You can have up to three modules:
  - EJB module containing entity, session and message beans – applicationName-ejb.jar
  - Client module containing the client application and GUI, and references to Session bean interfaces
  - Web module for a web front end - applicationName-ejb.war

Building a JEE application in Netbeans

Look at netbeans tutorial:
http://netbeans.org/kb/docs/javaee/javaee-entapp-ejb.html



Summary

- Services supported by an EJB container
- Entity Beans
- Session Beans
  - Stateful; stateless; singleton
  - Instance pooling and activation
  - Life cycles
- Message Beans
- Building a JEE application

Exam questions, Jan 2013/14

```
 @Stateless
public class CustomerFacade extends AbstractFacade<Customer> {

    @PersistenceContext(unitName = "CustApp-ejbPU")
    private EntityManager em;
    private Customer entityClass;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public CustomerFacade() {
        super(Customer.class);
    }
……
```

Answer the following questions based on extract of code above:

i) Is this code extract from an Entity Class, a Session Bean or a Message Bean? Explain the role of this type of bean as part of a Java EE application. **3 marks**

ii) Explain the annotation **@PersistenceContext**. **3 marks**

iii) Explain the annotation **@Stateless**. **3 marks**

iv) What services does an **Entity Manager** offer an Enterprise Java Bean? **4 marks**

Exam questions, Repeat 2013/14

a) Explain message beans under the following headings:

i) The role of a message bean in a Java EE application. **2 marks**
ii) How message beans differ from session beans. **4 marks**
iii) Message beans can avail of the same EJB container services as session beans. Apart from access to an entity manager, discuss two other services available to a message bean. **6 marks**

b) Read the extract of code below, which is from an **abstract façade** that session beans in a Java EE application extend. Answer the related questions following the code:

```
public abstract class AbstractFacade<T> {
private Class<T> entityClass;

public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;     }

protected abstract EntityManager getEntityManager();

public void create(T entity) {
getEntityManager().persist(entity);
}
public void edit(T entity) {
getEntityManager().merge(entity);
}

public void remove(T entity)
{getEntityManager().remove(getEntityManager().merge(entity));
}
public T find(Object id) {
return getEntityManager().find(entityClass, id);
```

i) Explain the role of the entity manager in the code above. **5 marks**
ii) There are four methods used above from the entity manager class: persist(), merge(), remove() and find(). Explain how each method effects the life cycle of an entity. **8 marks**

Exam Questions

January 2012, Q1 b)

- Discuss the range of services available to an Enterprise Java Bean via an EJB container, giving details of at least three such services. **10 marks**

January 2012, Q1 b)

- Enterprise Java Beans (EJB's) run within an EJB container such as that implemented in the Glassfish application server. Discuss the range of services that such an application server must provide to support the business logic on the server side of a JEE application. **12 marks**

Exam Questions

January 2011, Q2

a) There are three types of EJB's. Explain the role of each type of bean. **9 marks**

(Note: since 2011, entity classes are no longer consider to be EJBs)

b) Read the following business scenario and answer the related questions:

The library offers access to a range of online academic journals to all registered students. Students can log in using their student number and library pin. If login is successful, students can search for articles by journal, author, title and/or keyword. All articles matching the search criteria will be displayed. If a specific article is not available, the student can submit a request to the library to purchase the article. The request would be responded to a later date.

January 2011, Q2 contd.

(i) Identify the entity beans needed to implement the scenario described above. Suggest appropriate attributes for each entity bean. **3 marks**

(ii) Would message beans be useful in implementing any of the business logic? Explain your answer. **3 marks**

(iii) How many session beans would you recommend? For each session bean, give pseudo code for the business logic it would need to implement. (Note: you do not need to include linking to a persistence unit, or creating an entity manager.) For each query needed, decide if it should be a static or dynamic query, and justify your selection. **10 marks**