# COS30041 Creating Secure and Scalable Software

Lecture 06 – Stateful Session Bean (Business Object)

## Learning Objectives

- After studying the lecture material, you will be able to
  - Understand and describe what a stateful session bean is
  - Understand and explain the issues involved in programming stateful session bean
  - Distinguish the differences between stateless and stateful session beans
  - Program stateful session bean
  - Program client application that calls the services provided by stateful session beans

## Pre-requisites

- Stateless Session Bean

## Outline

- Stateful Session Bean
- Retaining Conversational State for Client
- Life Cycle of Stateful Session Bean
- Programming Stateful Session Bean

## Session Bean (Recap)

- A bean that models the business processes
  - Business logic
  - Business rules
  - Algorithms
  - Workflow
- Example
  - accessing bank account
  - verifying credit card details
  - preparing an invoice

## Lifetime of a Session Bean (Recap)

- The lifetime of a session or the lifetime of the client code that is calling the session bean
  - ☐ The time of a browser window is open
  - ☐ The time of your Java applet is running
  - ☐ A standalone client application is open
  - ☐ Another enterprise bean is using your session bean
- The EJB container will destroy session beans if clients time out

## General Issues (Recap)

- A Session Bean cannot be shared between clients
- Session beans do not represent data in a database
- Session beans are transaction aware

## Different types of Session Beans (Recap)

- Stateless session bean
- Stateful session bean

## Roadmap

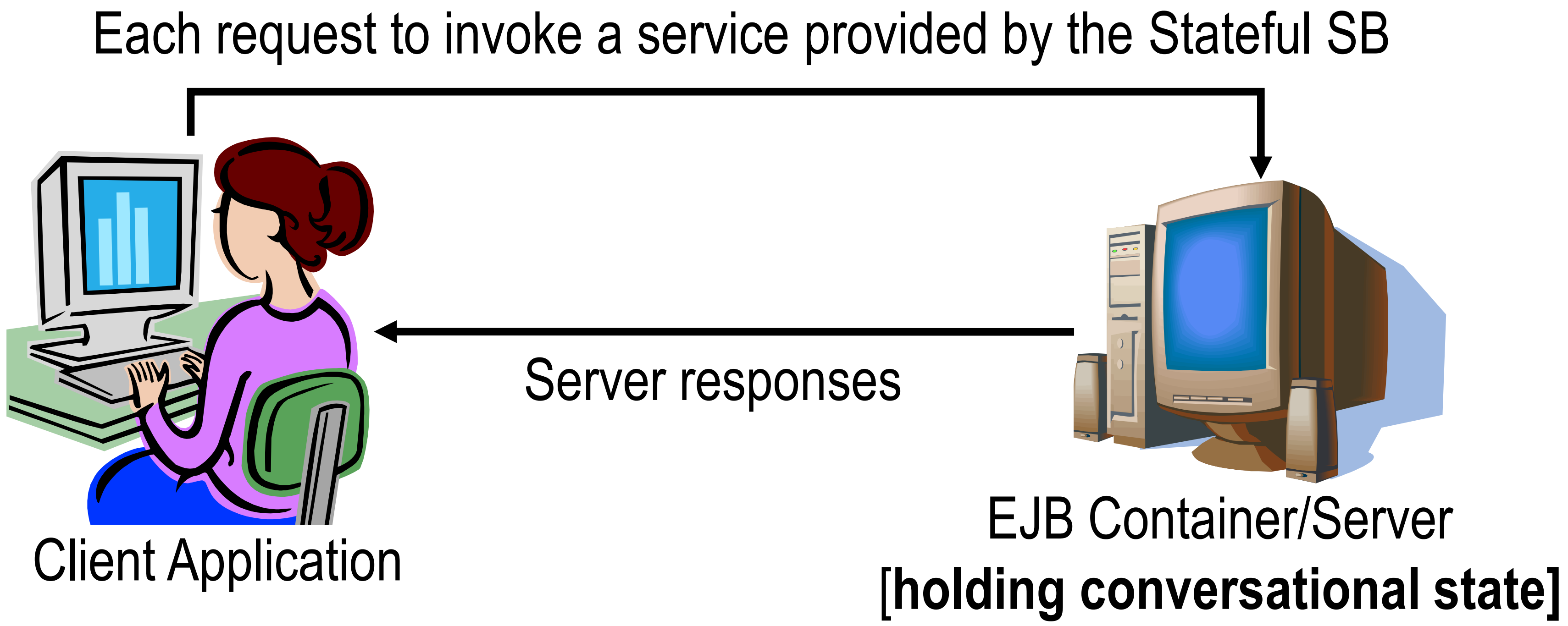
- **Stateful Session Bean**
- Retaining Conversational State for Client
- Life Cycle of Stateful Session Bean
- Programming Stateful Session Bean

## Stateful Session Bean

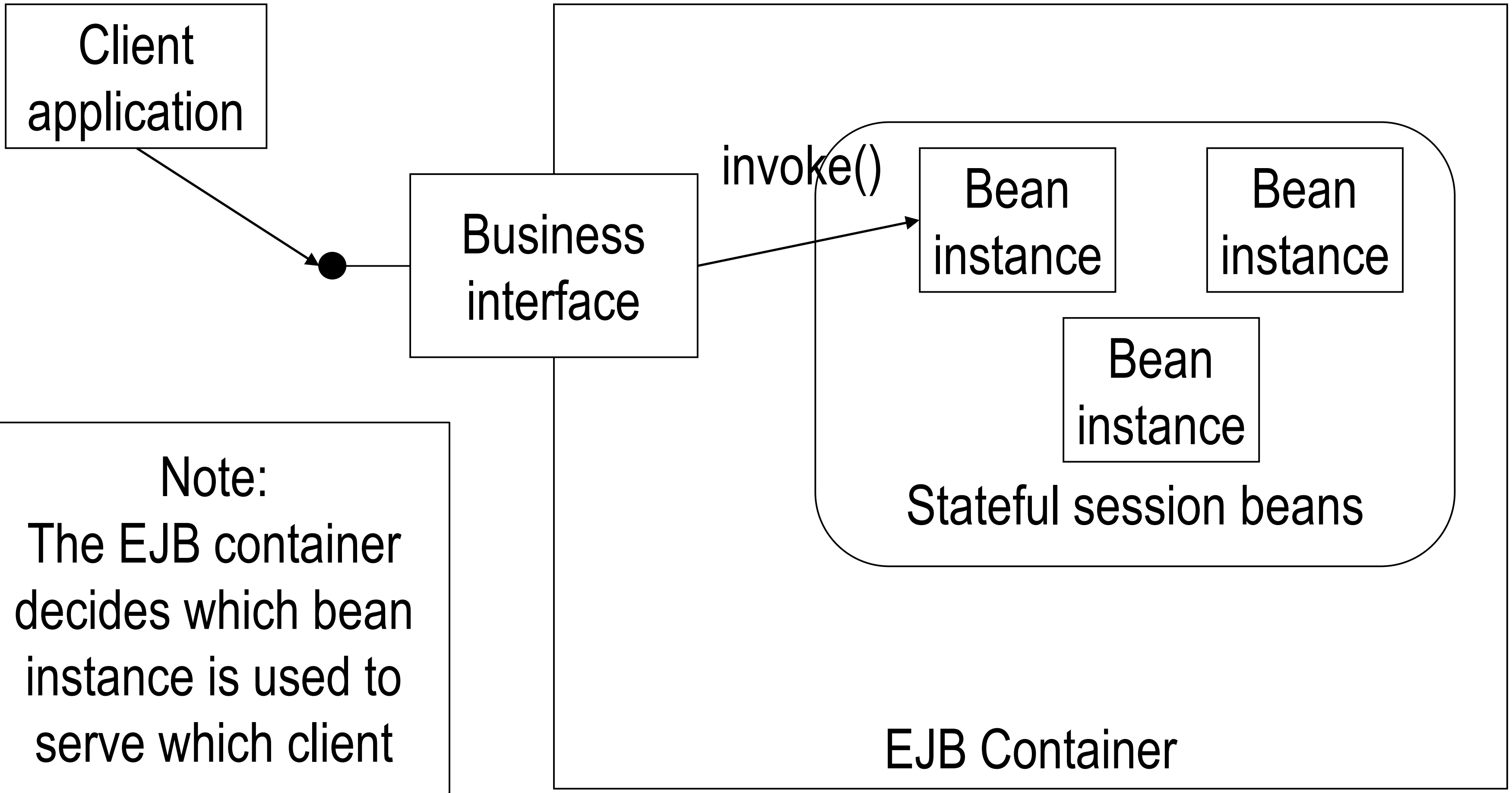
- A bean that is designed to service business processes that span ***multiple*** method requests (from client) or transactions
  - Each method request is considered as a conversation between the client and the bean
- Example: E-commerce Web store
  - Each time when user adds a product to the online shopping cart, it is a method request
  - The shopping cart is best modelled by a stateful session bean

# Stateful Session Bean – Typical Invocation

- The client application makes several requests (of a business process) to the EJB container/server
- The EJB container processes the requests and sends the results back to the client


Each request to invoke a service provided by the Stateful SB
Server responses
Client Application
EJB Container/Server
[holding conversational state]

# Stateful SB – Typical Invocation (cont'd)



Note:
The EJB container decides which bean instance is used to serve which client

Adapted from Fig. 4.1 of [MEJB,p.94]

## Why Stateful Session Bean?

- Need to model business processes where multiple requests are needed
- Multiple client interactions are needed
- General Recommendation: Avoid this if possible

## Stateful Session Bean – Avoiding it?

- Example: In real life banking, the teller may serve a client for a long time doing several transactions at a time
  - Check the balance
  - Deposit money to the account if balance is not enough
  - Transfer the money to another account
- We may then have a stateful session bean that hold the conversations with the client application (in the internet banking application)
- Problem: Too complicated. Avoid it. But how?

## Roadmap

- Stateful Session Bean
- **Retaining Conversational State for Client**
- Life Cycle of Stateful Session Bean
- Programming Stateful Session Bean

## Retaining Conversational State for Client

- Passivation
- Activation
- Through Passivation and Activation, the effect of “Bean Instance Pooling” with Stateful Session Beans can be achieved

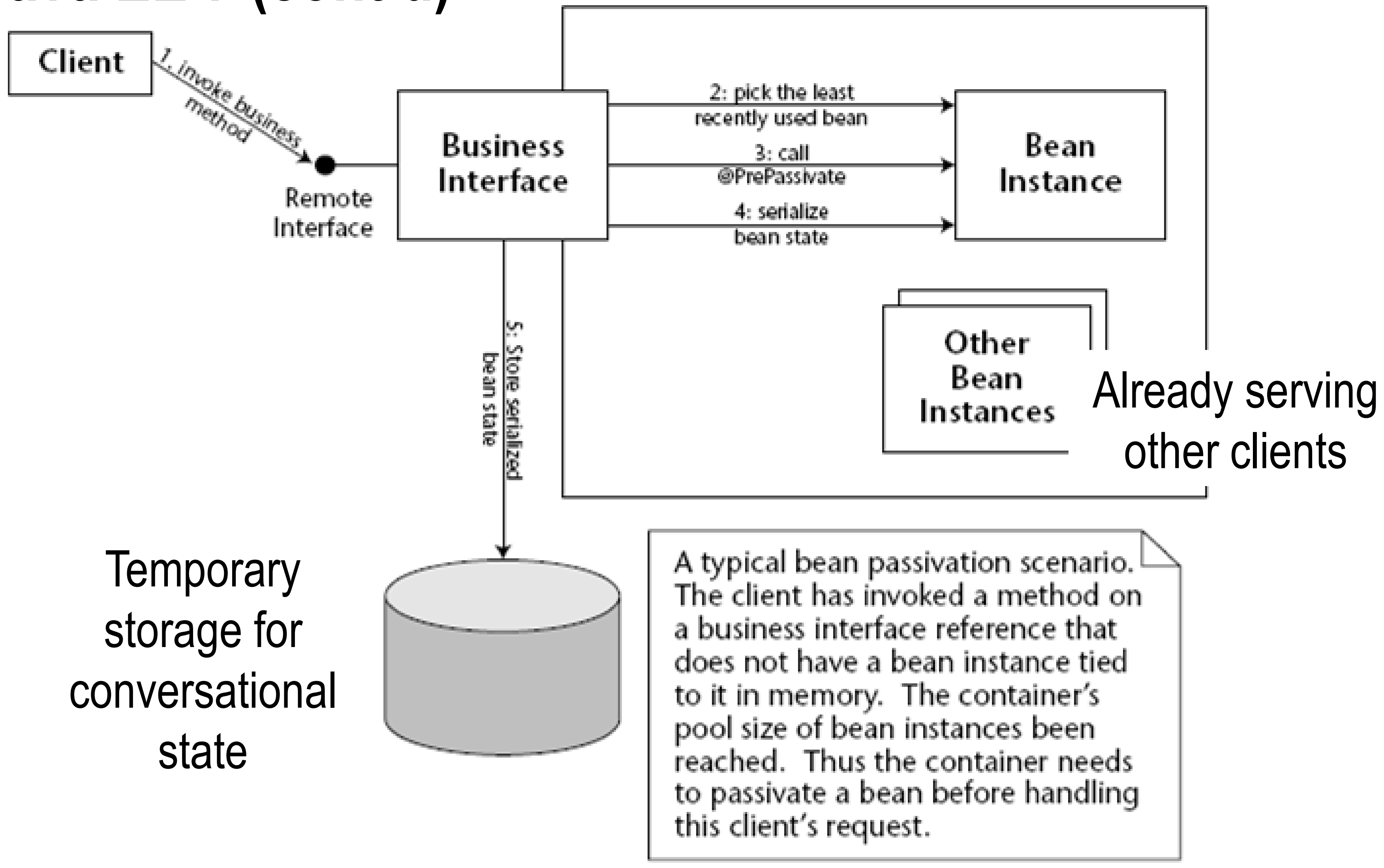
## Retaining Conversational State – Passivation

- Passivation
  - The process of the container swapping out a stateful session bean, saving its conversational state to a hard disk or other storage (temporarily)
- Decision to Passivate – Container specific
  - Mostly used is “The *Least Recently Used* (LRU)”
- It can occur at any time as long as the bean is not involved in a method call

## Retaining Conversational State – Passivation – Java EE 7

- The container will call the bean's optional PrePassivate callback method
  - defined using the `@PrePassivate` annotation
  - can only have one such method
- Purpose of running PrePassivate callback method
  - Release held resources (e.g. database connections and open sockets) that cannot be handled by the container
- The PrePassivate callback method is not needed if the bean does not hold any such resources

## Retaining Conversational State – Passivation – Java EE 7 (cont'd)


Client
1. invoke business method
Remote Interface
Business Interface
2: pick the least recently used bean
3: call @PrePassivate
4: serialize bean state
Bean Instance
Other Bean Instances
Already serving other clients
5: Store serialized bean state
Temporary storage for conversational state

A typical bean passivation scenario. The client has invoked a method on a business interface reference that does not have a bean instance tied to it in memory. The container's pool size of bean instances been reached. Thus the container needs to passivate a bean before handling this client's request.

Note: Adapted from Fig. 4.2 of [MEJB3,p.99]

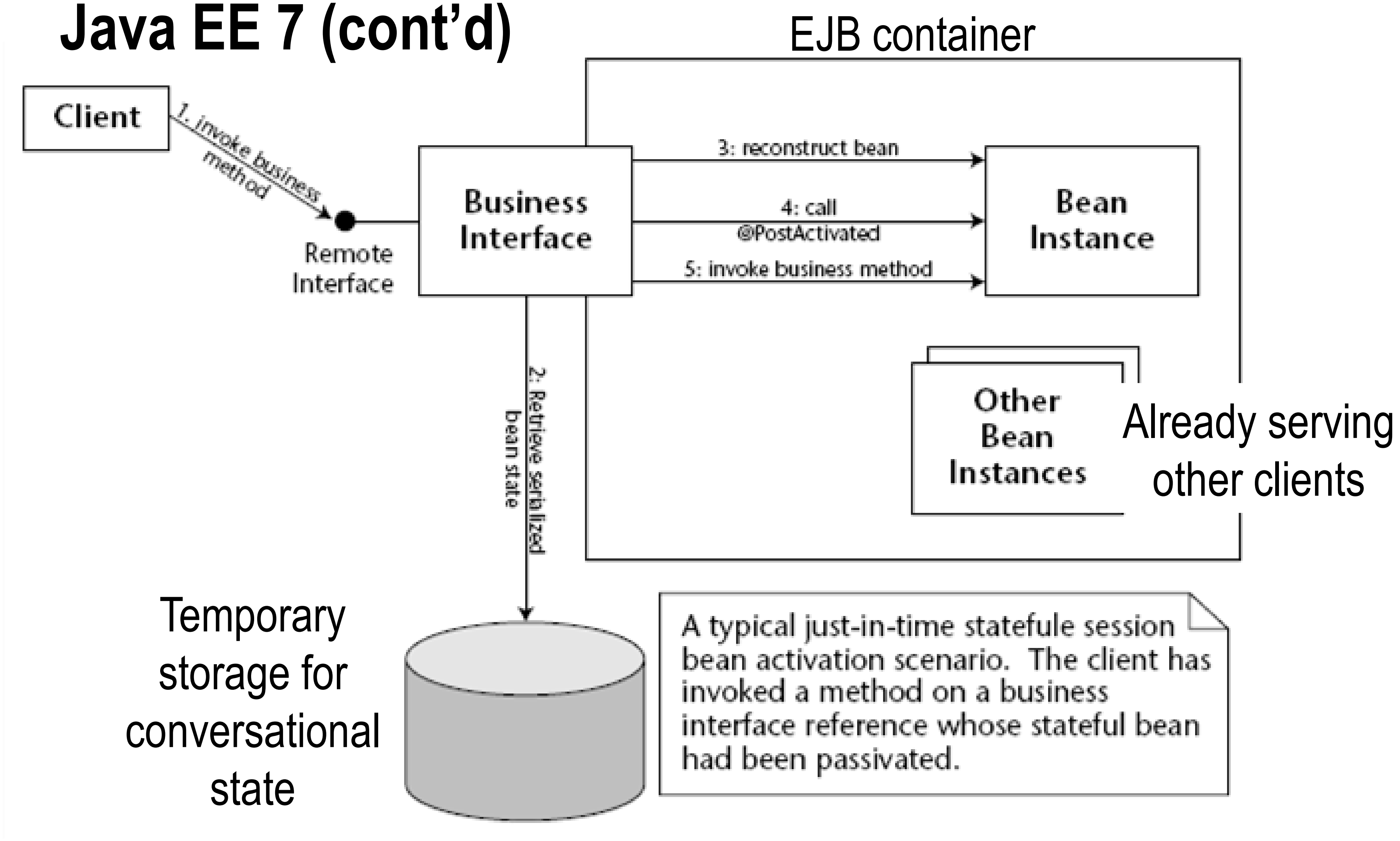
## Retaining Conversational State – Activation

- Activation
  - The process of the container swapping in a stateful session bean, reading its conversational state from the hard disk or other storage
- Decision to Activate – Container-specific
  - Mostly used is “*Just-in-time*” strategy

## Retaining Conversational State – Activation – Java EE 7

- The container will call the bean's optional PostActivate callback method
  - defined using the `@PostActivate` annotation
  - can only have one such method
- Purpose of running the PostActivate callback method
  - Restore any resources previously released during Passivation (e.g. database connections and open sockets)
- The PostActivate callback method is not needed if the bean does not hold any such resources

# Retaining Conversational State – Activation – Java EE 7 (cont'd)


EJB container
Client
1. invoke business method
Remote Interface
Business Interface
3: reconstruct bean
4: call @PostActivated
5: invoke business method
Bean Instance
2: Retrieve serialized bean state
Other Bean Instances
Already serving other clients
Temporary storage for conversational state

A typical just-in-time statefule session bean activation scenario. The client has invoked a method on a business interface reference whose stateful bean had been passivated.

Note: Adapted from Fig. 4.3 of [MEJB3,p.99]

## Roadmap

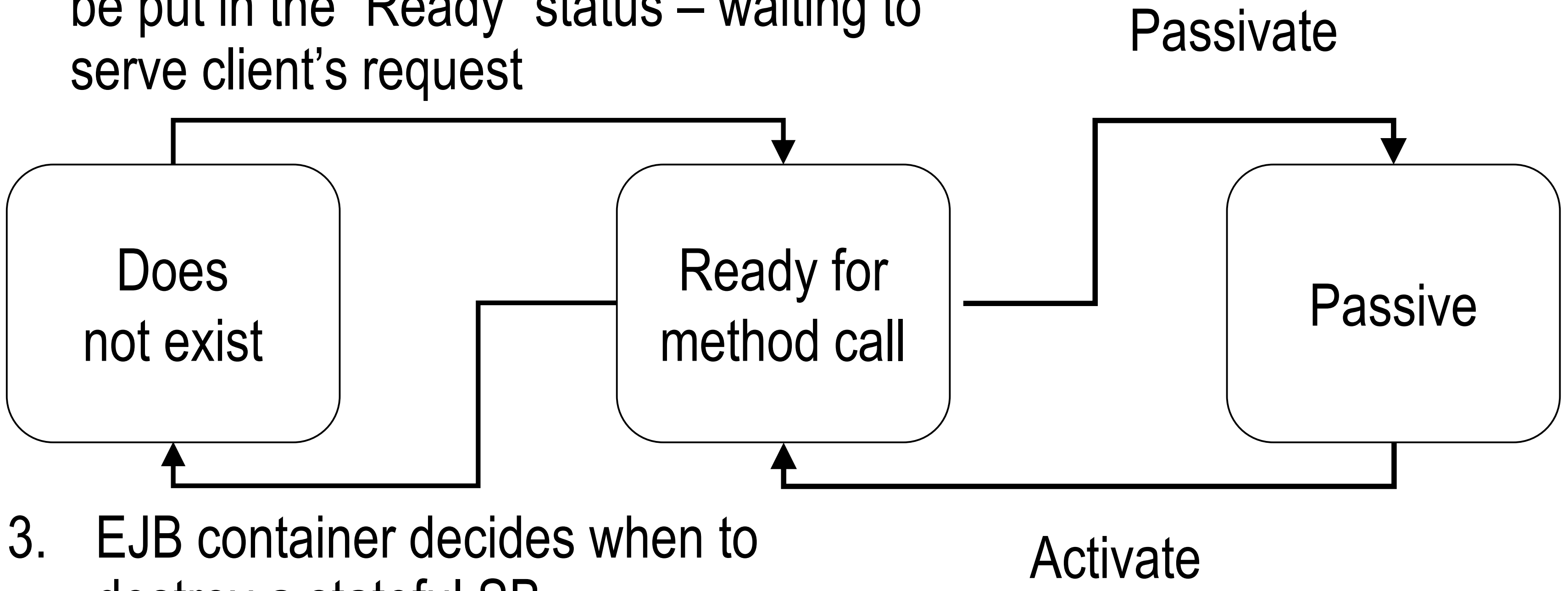
- Stateful Session Bean
- Retaining Conversational State for Client
- **Life Cycle of Stateful Session Bean**
- Programming Stateful Session Bean

## Life Cycle of Stateful Session Bean

- Similar to the life cycle of a stateless session bean except that there is a *passive* state
- Possible states
  - "Does not exist"
  - "Ready for method call"
  - "Passive"

## Life Cycle of Stateful Session Bean (cont'd)

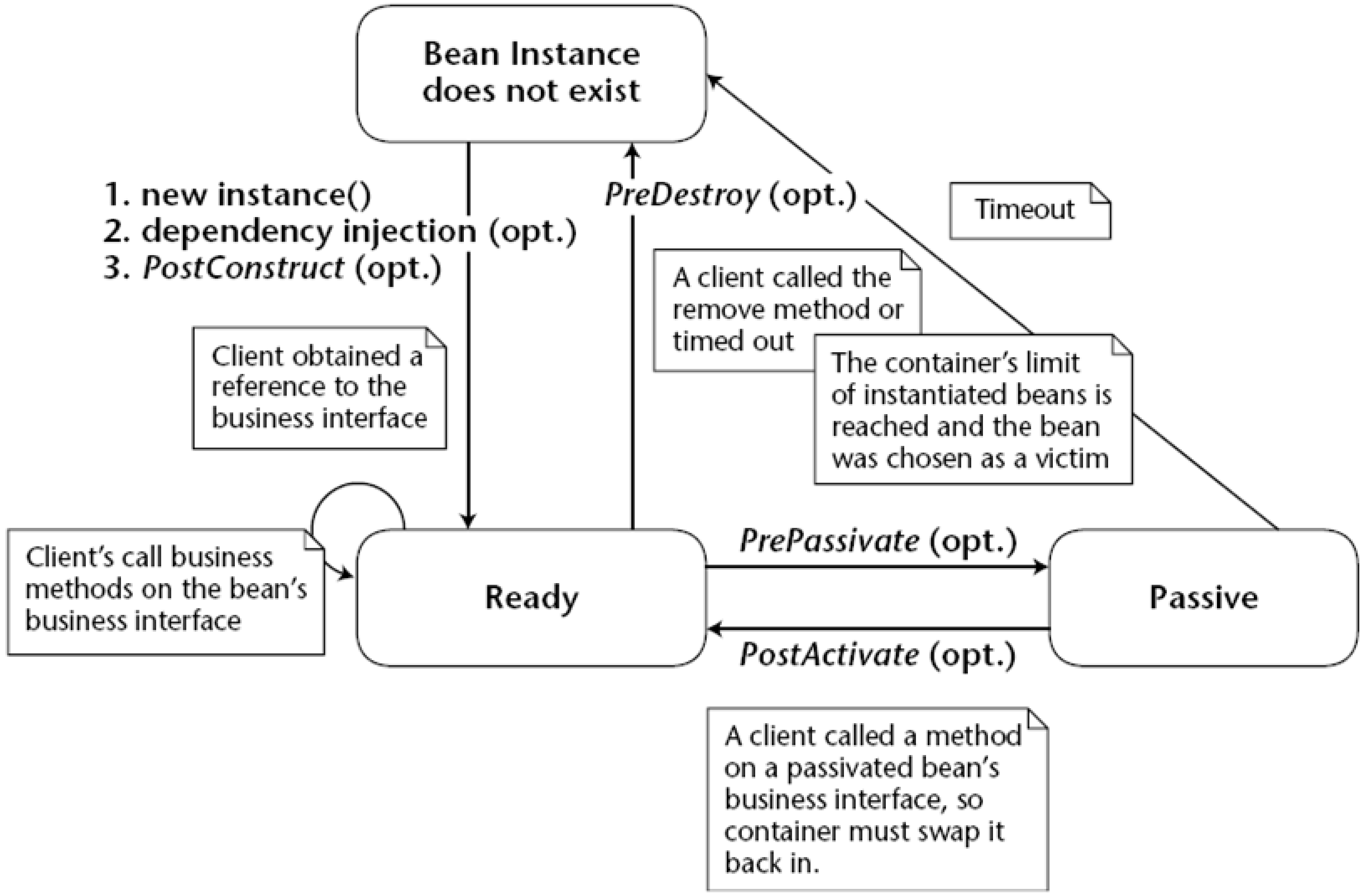
1. EJB container decides when to create a stateful SB
2. Once created, the stateful SB instance will be put in the "Ready" status – waiting to serve client's request



3. EJB container decides when to destroy a stateful SB
4. Once destroyed, the stateful SB instance does not exist

Note: Adapted from Fig. 20.3 of [JEE5T,p.644]

# Life Cycle of Stateful Session Bean (cont'd)


Bean Instance does not exist
1. new instance()
2. dependency injection (opt.)
3. *PostConstruct* (opt.)
*PreDestroy* (opt.)
Timeout
A client called the remove method or timed out
Client obtained a reference to the business interface
The container's limit of instantiated beans is reached and the bean was chosen as a victim
Client's call business methods on the bean's business interface
Ready
*PrePassivate* (opt.)
Passive
*PostActivate* (opt.)
A client called a method on a passivated bean's business interface, so container must swap it back in.

Note: From Fig. 4.5 of [MEJB3,p.113]

## Roadmap

- Stateful Session Bean
- Retaining Conversational State for Client
- Life Cycle of Stateful Session Bean
- **Programming Stateful Session Bean**

## Programming Stateful Session Bean

**The Client Side**

Some programs that request the services of a stateful session bean

- A client application
- A web page (e.g. JSFs)

**The Server Side**

- Stateful session bean (EJB)

## Prog. Stateful SB – Server – Java EE 7 specific

- The Remote Interface
- The Local Interface
- The Bean Class
- The Deployment Descriptor
- The Vendor-Specific Files
- The EJB-JAR File

[From NB7.1] need to put this in a separate package outside the EAR / EJB-JAR

Done by Sun's Application Server or some IDE

## Prog. Stateful SB – Server – Java EE 5/6/7 (cont'd)

| Naming Convention – Stateful Session Bean | | | |
|---|---|---|---|
| | JEExT / MEJB | NB6.9.1 or later (key in) | NB6.7.1 (key in) |
| Name of EJB | SBean (CountBean) | SBean (CountBean) | S (Count) |
| The Bean class (omitting ".java" ext) | SBean.java (CountBean) | SBean.java (CountBean) | SBean.java (CountBean) |
| The Remote Interface class (omitting ".java" ext) | S.java (Count) | SBeanRemote.java (CountBeanRemote) | SRemote.java (CountRemote) |
| The Local Interface class (omitting ".java" ext) | SLocal.java (CountLocal) | SBeanLocal.java (CountBeanLocal) | SLocal.java (CountLocal) |

## Programming the Remote Interface – Java EE 5/6

- A simple Plain Old Java Interface (POJI)
- Use the annotation "`@Remote`" to indicate the interface class is a remote interface for the EJB
  - Example: ([NB691 or later])
    ```
    @Remote
    public interface CountBeanRemote { … }
    ```
- Expose every business method about the stateful session bean for remote client applications
  - Name the methods and their corresponding parameters
  - Example
    ```
    public int count();
    ```

## Programming the Local Interface – Java EE 5/6

- A simple POJI
- Use the annotation "`@Local`" to indicate the interface class is a remote interface for the EJB
  - Example: ([NB691 or later])

```
@Local
public interface CountBeanLocal { … }
```

- Expose every business method about the stateful session bean for local client applications
  - Name the methods and their corresponding parameters
  - Example

```
public int count();
```

## Programming the Bean class – Java EE 5/6/7

- A simple Plain Old Java Object (POJO) implementing the business methods
- Use "`@Stateful`" to indicate that it is a Stateful SB
- [Optional in NB7.1] Use "`@Remote(…)` / `@Local(…)`" to indicate the corresponding remote / local interface class
- Example: ([JEExT])

Optional in NB7.1

```
@Stateful
@Remote(Count.class)
@Local(CountLocal.class)
public class CountBean { … }
```

## Prog. the Bean class – Java EE 5/6/7 (cont’d)

- Program the business methods as defined in Remote and Local Interfaces
  - Example

```
public int count() {
 counter++;
 …
}
```

## Prog. the Bean class – Java EE 5/6/7 (cont'd)

- [Stateful] Program the **optional** life cycle callback methods
  - `@PostConstruct`
  - `@PreDestroy`
  - `@PostActivate`
  - `@PrePassivate`
- [Stateful] Program the **optional** business method annotated `@Remove` – for removing bean instances

## Prog. the Bean class – Java EE 5/6/7 (cont'd)

The Life Cycle Callback methods

- Methods called by EJB container to maintain the life cycle (status) of the Stateful SB
- The `@PostConstruct` method
  - called after a new bean instance is created and before any business methods are called
  - Example

```
@PostConstruct
public void construct(InvocationContext ctx) {
 …
}
```

## Prog. the Bean class – Java EE 5/6/7 (cont'd)

The Life Cycle Callback methods (cont'd)

- The `@PreDestroy` method
  - called after the `@Remove` method has completed and before the container removes a bean instance
  - Example

```
@PreDestroy
public void destroy(InvocationContext ctx) {
 …
}
```

## Prog. the Bean class – Java EE 5/6/7 (cont'd)

The Life Cycle Callback methods (cont'd)

- The `@PostActivate` method
  - called after the bean instance is activated from its "passive" state
  - Example

```
@PostActivate
public void activate(InvocationContext ctx) {
 …
}
```

## Prog. the Bean class – Java EE 5/6/7 (cont’d)

The Life Cycle Callback methods (cont’d)

- The `@PrePassivate` method
  - called before the bean instance enters the "passive" state
  - Example

```
@PrePassivate
public void passviate(InvocationContext ctx) {
 …
}
```

## Prog. the Bean class – Java EE 5/6/7 (cont’d)

The Life Cycle Callback methods (cont’d)

- The `@Remove` method
  - called by EJB container before removing the stateful session bean instance

## Programming the Stateful SB – Example

- WANT: a Stateful Session Bean "`CountBean`" to
  - greet user with `userName` (default is "`World`") and
  - count how many times the user requested a particular method in the bean
- NEED two methods:
  - getGreetings(String userName) – returns the greetings
  - count() – updates the number of times the user makes a request
- ISSUE:
  Where to call "count()"? In CountBean? In CountClient?

## Prog. SFSB – Example (Java EE 5/6)

Example: ([NB691 or later])

- The Remote Interface
  - See EJ-Session-Stateful demo – `CountBeanRemote.java`
- The Local Interface
  - See EJ-Session-Stateful demo – `CountBeanLocal.java`
- The Stateful Session Bean Class
  - See EJ-Session-Stateful demo – `CountBean.java`

## Programming Stateful SB – Client

- The application client
  - An application that
    calls the business methods of an EJB object remotely and then displays the returned results locally
  - When compiling client application in a different machine,
    you need the EJB-JAR file from the EJB developed on the server

## Programming Stateful SB – Client

- Naming Convention: Stateful Session EJB - SBean
  - The Client Application – SClient ([JEExT])
  - NetBeans uses “Main.java” but I renamed it to “SClient.java” for consistency purposes (?) in the demo code
- Example: Stateful Session EJB – CountBean
  - The Client Application - CountClient

## Prog. the Client for SFSB – Java EE 5/6

- Use "`@EJB private S s;`" ([JEExT]) to declare the required stateful session EJB – "`SBean`" – as a variable whose name is "`s`" in the client application
- Example: ([JEExT])

```
@EJB
private static Count count;
```

- Call the business methods of the EJB as usual
- Example:

```
Count.count();
```

- Remove the bean, if needed

## Prog. the Client for Stateful SB – Example

Example: ([NB691 or later])

- WANT: a client to call the business methods provided by the "`CountBean`" stateful session EJB
- Client Application
  - See EJ-Session-Stateful demo – `CountClient.java`

## Deploying the EJB

- This involves
  - Preparing the deployment descriptor,
  - Preparing any vendor-specific files, and
  - Packaging the EJB-JAR file
- NetBeans IDE handles these steps automatically
  - Sun’s Application Server can only prepare Sun-Specific files
  - Need to consult the corresponding vendors for their specific files

## Running the Client Application

- This involves
  - deploying the EJB services on an EJB container/server, and
  - executing the Client Application
- See Lab Sheet for detail

## Bean Instance Pooling – Stateful SB?

- Using Activation and Passivation

## References

- [MEJB3] R.P. Sriganesh, G. Brose, M. Silverman (2008) *Mastering Enterprise JavaBeans 3.0*, 4th ed., John Wiley & Sons
  - Chapter 4
- [JEE7T] E. Jendrock et al. (2014) *The Java EE 7 Tutorial*, Oracle, August 2014
  - Chapters 32 – 34