EJB Session Beans

JPA Slides #4

Software Component

No commonly accepted definition

Our (informal) definition

- Self-contained, reusable software unit that can be integrated into an application

The component model for JEE is the Enterprise JavaBeans model (focus on business services)

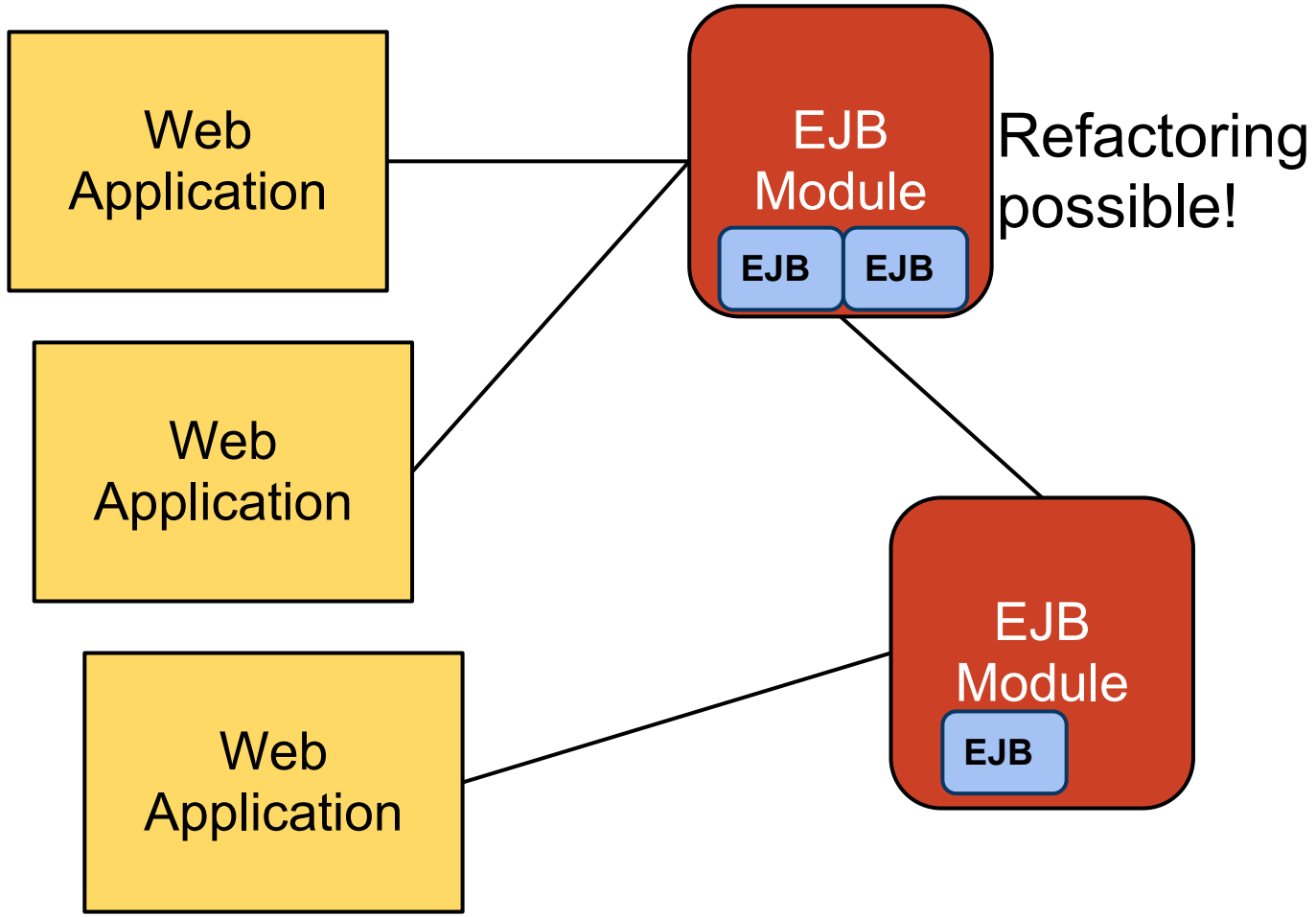
- Aka EJB's (which are classes)
- EJBs are default transactional (we don't need to handle, see code samples)

EJB Modules

Javas proposal for modular enterprise applications
- EJB collected in EJB modules (jar-files)
- **Modules** are the (reusable/refactored) components
- Modules must run in EJB container (GlassFish or other)

Specification EJB 3.1
- JSR 318 (10 dec. 2009)
- Also a "light" version (implementors don't have to support full EJB specification). Possible to use EJB's in pure Web applications

JEE Architecture



Services For EJBs

Added services/possibilities for EJB classes

- Component life cycle
- Dependency injection (inferior to CDI)
- Container managed transactions
- Concurrency handling
  "…The container serializes calls to each stateful and stateless session bean instance [EJB]. Most containers will support many instances of a session bean executing concurrently; however, each instance sees only a serialized sequence of method calls. Therefore, a stateful or stateless session bean does not have to be coded as reentrant [~thread safe]…"
- Aspect oriented programming, interceptors
- Security
- Scheduling
- Web service endpoint
- Messaging
- …

Restrictions on EJBs (and helper classes)

-Not use the java.lang.reflect Java Reflection API to access information unavailable by way of the security rules of the Java runtime environment
- Not read or write non-final static fields
- Not use this to refer to the instance in a method parameter or result
- Not use the java.awt/swing packages to create a user interface
- Not create or modify class loaders and security managers
- Not redirect input, output, and error streams
- Not obtain security policy information for a code source
- Not create or manage threads
- Not use thread synchronization primitives to synchronize access with other enterprise bean instances

...

...

- Not.....stop the Java virtual machine....

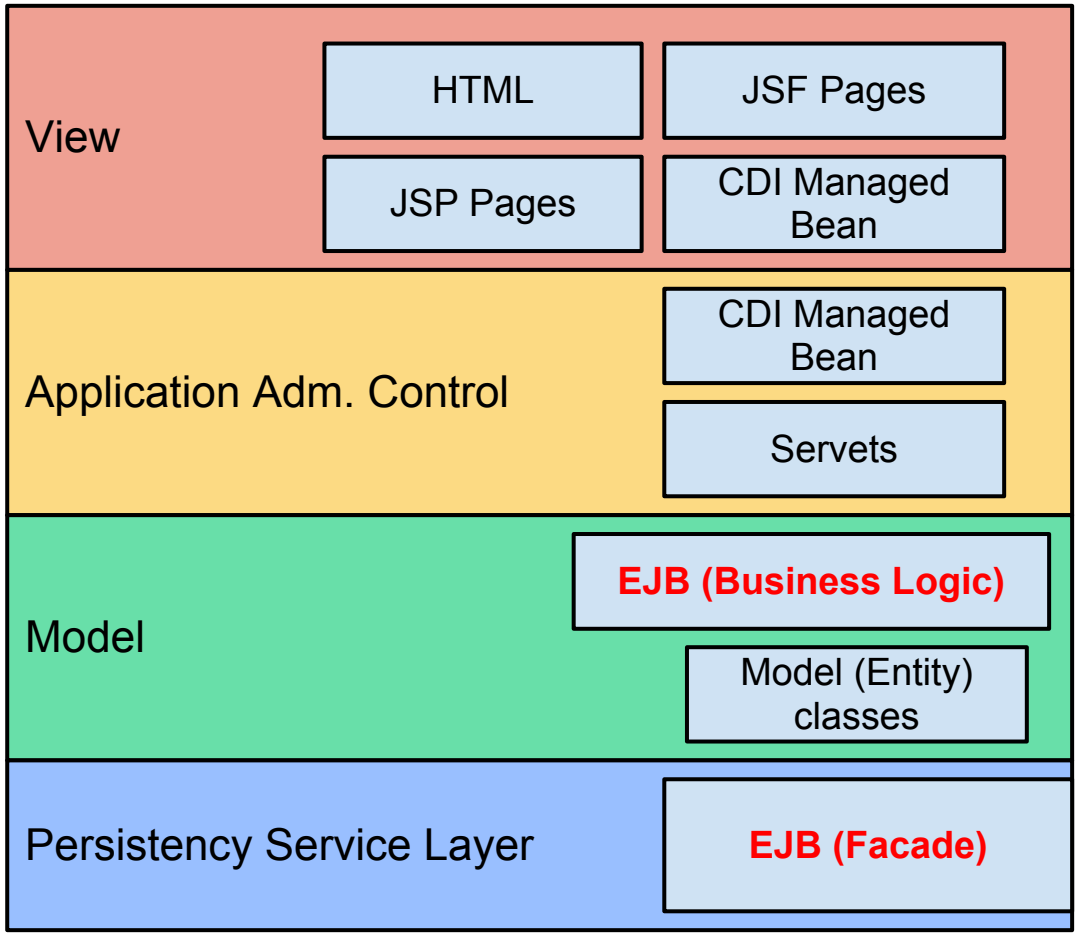
SUMMARY: Use as intended!

Types and Purpose of EJBs

Session Beans (SB)
- Facade to persistence, DAO
- Also business logic, processes, work flows

Message Driven Beans (MDB)
- Used by Java Message Service (JMS). Asynchronous message service with high quality of service
- We don't use...

Location of EJBs



Session Beans

Executes on behalf of a unique client

- The client running the session (i.e session scoped)
- Short lived, will not survive a server crash

Transaction-aware

- Transactions as needed, handled by container

Further divided into

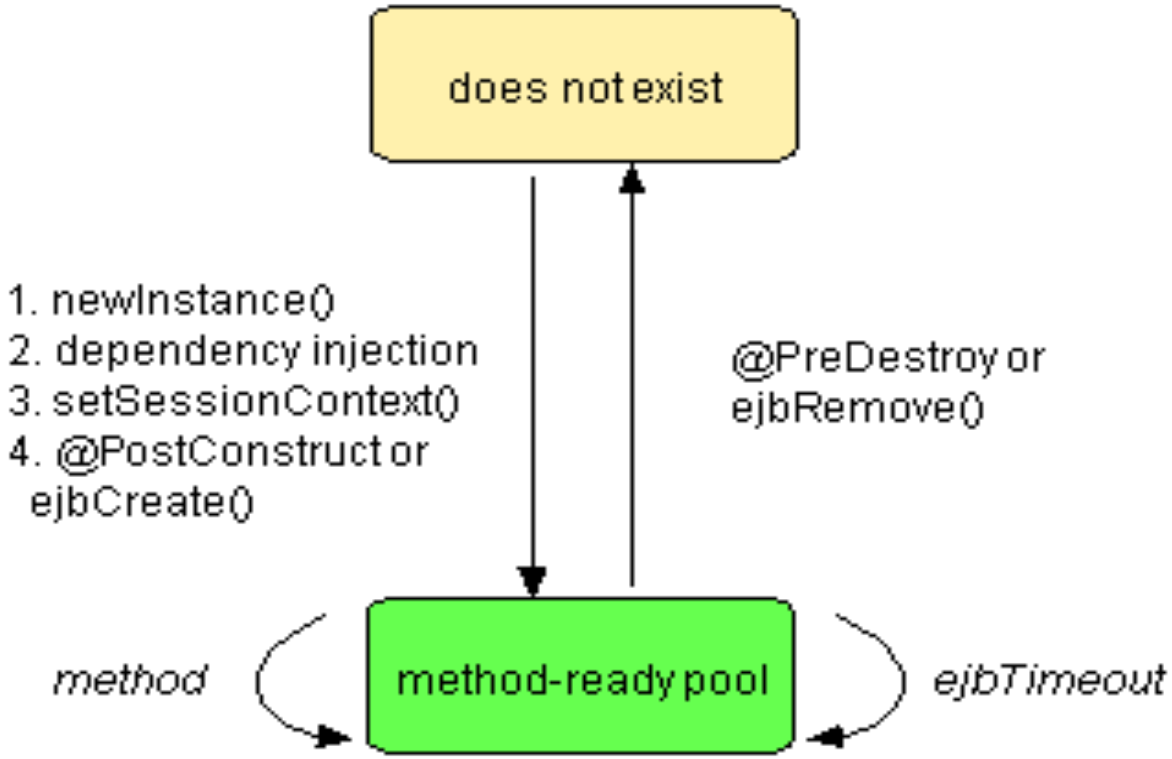
- Stateless
- Stateful
- Singleton: Singleton pattern (NOT same as CDI Singleton)
- Variations: Local bean (in same JVM), Remote bean (possible other JVM)

Stateless SB

No conversational state (no instance variables)
- Task possible to conclude in single method call
- Efficient, share by many clients
- Pooled by container
- Can implement a Web Service

The preferred type to use
- We always strive for statelessness

Stateless SB Life Cycle



Annotations for EJB callback methods
@PostConstruct
@PreDestroy
@PrePassivate
@PostActivate
@Init
@Remove

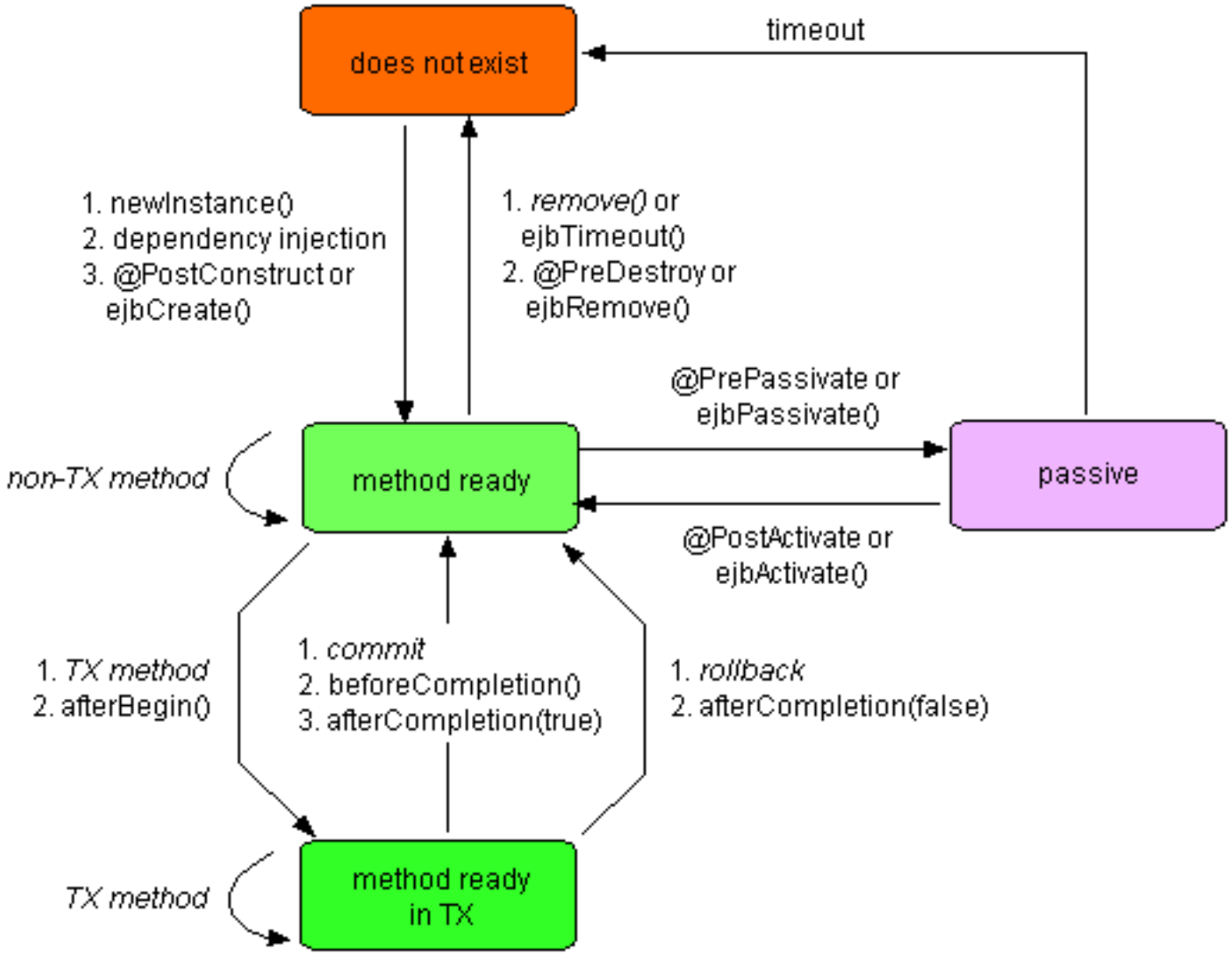
Stateful SB

Conversational state (instance variables)
- Task done in several steps
- ShoppingCart, add items...
- Container can't share (less efficient compared to stateless)
- State retained for the session
- Passivated/Activated (moved out/in memory)

Complex
- Conversational state not transactional (rollbacks could lead to inconsistency)
- Have to handle in code

Normally avoid

Stateful SB LifeCycle



EJB Singleton SB

Instantiated once per application
- Possible use : “the database”, “printer”,...
- Shared by many clients
- Maintain state between client invocations
- Concurrency issues: Container Managed (@Lock), bean Managed (@ConcurrencyManagement)...or not allowed (exception if...)

Initialization (@Startup)
- Dependencies between Singleton → initialization order matters (@DependsOn)

NOT SAME AS CDI Singleton

Session Beans and Interfaces

Possible to use Session Beans without interface
- No-interface view, @LocalBean
- Possible for bean to implement a local interface with @Local annotation
- ...or a remote interface with @Remote annotation
- Possible implement both

Term: **"Business interface"**, methods visible to client (other bean, ...)

Local Interface

Same as no interface, but we should of course program to interfaces for SE reasons, i.e. use local interface! (not always in sample code)

- Client end SB in same JVM
- Call by reference
- Inexpensive

Remote Interface

Remote calls: Using CORBA or RMI to call directly between Java objects no HTTP), we don't ...

Remote beans must implement a remote interface

Client end SB in different JVMs
- Possible different servers
- Call by value
- Parameters/return values must be serializable
- Calls potentially expensive, network...
- Container must support CORBA/IIOP and Java RMI

Asynchronous requests

Bean calls default synchronous

Asynchronous request possible
- Long running methods
- Send mail, search database, ...
- Method annotation: @Asynchronous
- If annotation on class, all methods asynchronous
- IFuture<T> as return type (or void)

Calls: Beans and Non Beans

SB calling SB: OK

SB calling non bean (i.e. using some service): OK

Non bean calling SB
- Avoid
- Not possible with dependency injection in non beans, possible to fix with JNDI possible but tedious
- Also transactions issues etc.

Resources to Inject into EJB's

Resources	Stateless	Stateful	MDB	Interceptors
JDBC DataSource	Yes	Yes	Yes	Yes
JMS Destinations, Connection Factories	Yes	Yes	Yes	Yes
Mail Resources	Yes	Yes	Yes	Yes
UserTransaction	Yes	Yes	Yes	Yes
Environment Entries	Yes	Yes	Yes	Yes
EJBContext	Yes	Yes	Yes	No
Timer Service	Yes	No	Yes	No
Web Service reference	Yes	Yes	Yes	Yes
EnityManager, EntityManagerFactory	Yes	Yes	Yes	Yes

And also inject one EJB into another

Injection annotations in EJB

Some samples...

```
@PersistenceUnit(s), to get an EntityManagerFactory

@PersistenceContext(s), to get an EntityManager

@EJB(s), to get other enterprise beans

@WebServiceRef(s), to get a web service reference
```

Inject EJBs into CDI's

Possible to inject EJBs in various other parts of application

```
@Named("backingbean")    // CDI Bean
@RequestScoped
public class MyBackingBean {
    @EJB                          // Inject EJB
    MyEJB myEjb;
```

Use: NetBeans > Insert Code ... > Call Enterprise Bean

If injection not possible have to use Java Naming and Directory Interface (JNDI), lookup (beans automatically will be assigned JNDI names), avoid (hard to figure out the name of bean)

CDI Scopes and EJBs

CDI scopes handled automagically (using proxies), should be no problem

Timer Service

Used for long lived business processes
- Calendar based
- Send an email: “Time for Christmas ... buy, buy, buy”..

Setup
- Let EJB have reference to Timer
- EJB's define and schedule callback methods using @Schedule, @Schedules

Container call back at scheduled times

A **persistent timer** will survive server crash

Getting a Timer

Automatic, use ...

- Each @Schedule annotation corresponds to single persistent timer
- Possible to create non-persistent

Programmatic, avoid ...

- Need object implementing TimeService interface
- Object injected as @Resource...
- Must provide @Timeout annotated method, called at timer timeout

EJB Interceptors

Aspect Oriented Programming (AOP)
- Cross cutting concerns, remember Filter

JEE interceptors used for AOP

Types of interceptors
- Life cycle interceptors
- Method interceptors
- ...much of this also in CDI but we don't use

EJB Method Interceptors

@AroundInvoke on a bean method

- Method called when any other bean method entered or exited (not very cross cutting)
- Some restrictions on method

Signature

- Object name(InvocationContext ic) throws Exception
- Not final or static

Possible to create interceptor class

- With the interception method
- Probably better: Cross cutting

EJB as RESTful End Points

Possible, ..

```
@Stateless  // An EJB
public class MyResourceBean {

    @Context
    private UriInfo ui;
    @EJB
    OtherBean otherBean;

    @GET  // JAX-RS
    @Produces({"application/xml", "text/html"})
    public List<Message> getMessages() {...}

    @POST
    public Response addMessage(String msg) throws URISyntaxException
    {...}

}
```

EJB Testing

Has been a major disadvantage, must run in container

But ... can use embedded container (seems a bit slooooooow...)

- Possible to test session beans with JUnit test
- NetBeans: Select bean > Tools > Create Unit Tests

… Now Continue…

.. with the Managing Persistent Objects slides (#3)