Enterprise JavaBeans

Enterprise JavaBeans

Enterprise JavaBeans is a server-side software component that encapsulates business logic of an application.

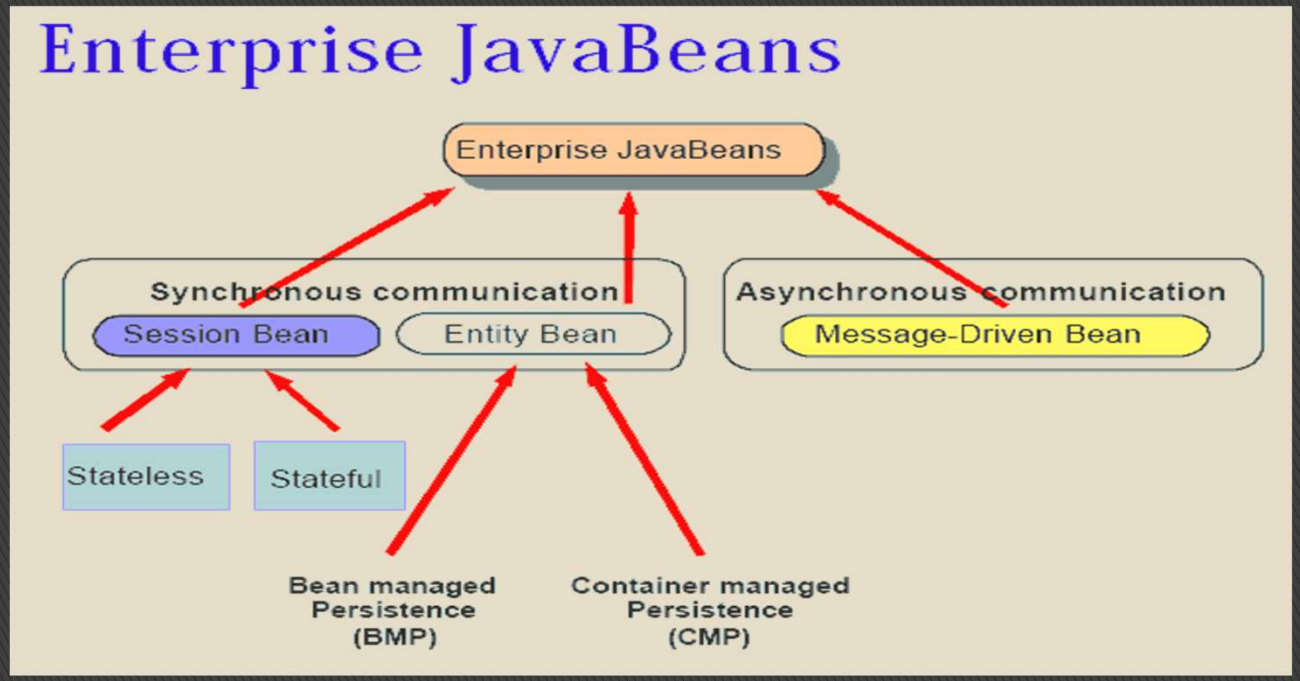
Enterprise JavaBeans

They run in an EJB **container** which provides the following services:

- loadbalancing, failover
- clustering support
- transparent network communication
- transparent concurrency management (Executor Service)
- transparent transactions
- other services: security, mailing, timer, batch execution
- EJB 3.2 (Java EE 7)

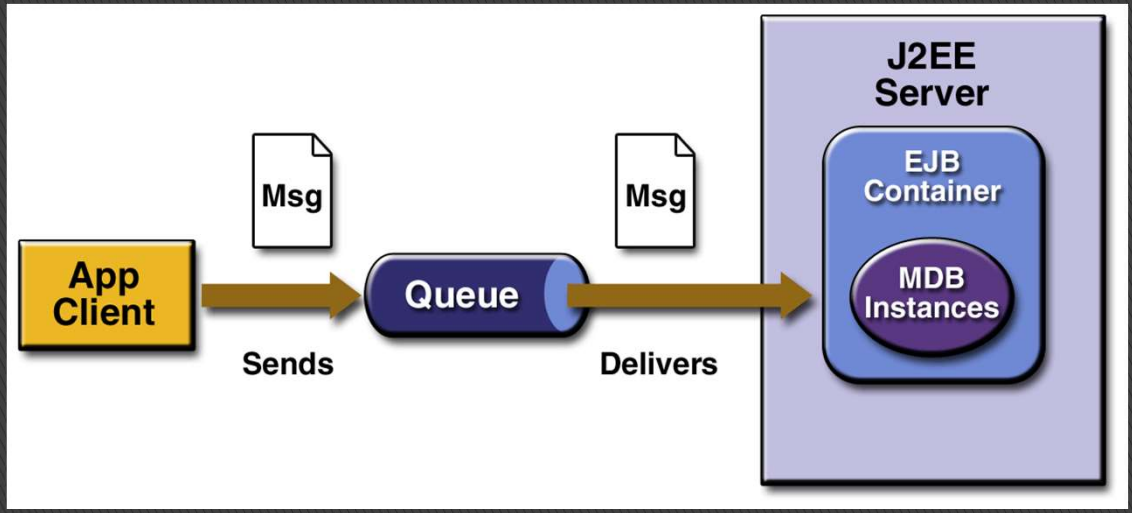
EJB types

- Session Bean
- Message-driven Bean(from EJB 2.0)
- (Entity Bean) – pruned JPA instead



Message Driven Bean

- Asynchronous message handling
- Enterprise messaging systems
  - IBM WebSphere MQ (earlier IBM MQSeries)
  - Sun Java System Message Queue
  - HornetQ – JBOSS



Session Beans

- they represent business processes (webshop)
- Use cases correspond to methods and methods are grouped into beans
  - methods: checkCreditCardBalance, payItem
  - bean: PaymentBean
- Scalability, load balancing

EJB3 Session Bean structure

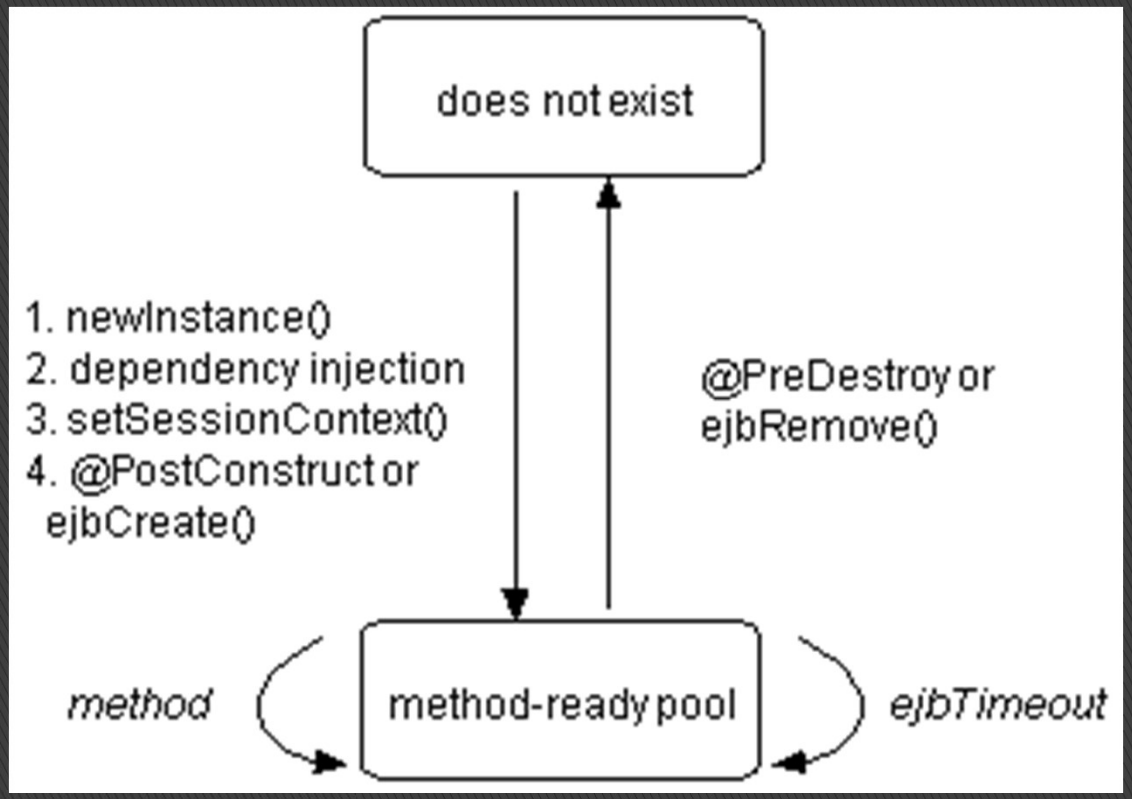
- **Enterprise Bean class (implementation class)**
- **Business interface**
  - **Remote interface – @Remote**
    Can be accessed from remote JVM–s, serialization overhead + network overhead
  - **Local interface – @Local**
    Can only be accessed on the same server, parameters are passed by reference, no serialization.
  - **No–interface view –** only used locally
- **Wrapper class generated by container**
  - Implements the business interface
  - Adds middleware functionality (transaction, security…) and delegates client calls

Types of Session Beans

- Stateful (has state – data saved to bean will be accessible during the whole user session)
- Stateless (clients may receive a new bean instance for every call – data save in stateless beans may be not accessible during the next call)
- Singleton

Stateless session Bean

- @Stateless annotation



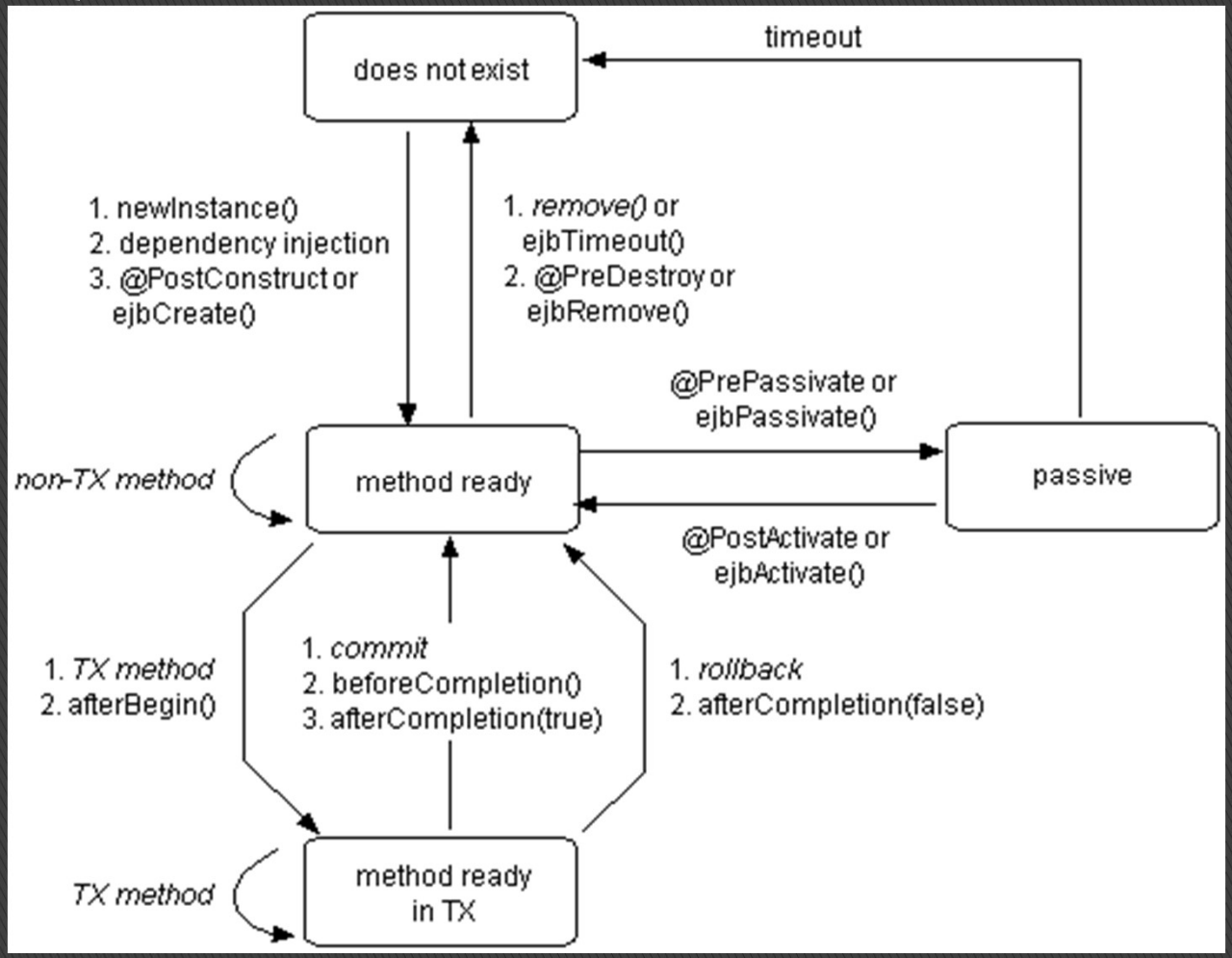
Pros of using Stateless session bean

- Pooling: the bean pool holds instances of the session bean and returns an instance from the pool after a client call
- Scalable – because there is no state any free instance from the pool can be returned
- No clustering overhead – if client is redirected to other cluster node it will be served from the pool on that node
- Easy transaction management – *ContainerManaged vs BeanManaged*
- Remote calls – access other JVM over network

Stateful session beans(@Stateful)

- Classical use case: online webshop cart
- If we have a lot of clients and each client needs its bean instance –> out of memory errors
- Bean state may be saved to hard drive (passivation) after long time of non-usage
- client calls passivated bean: has to be activated (read into memory)
- saving the state is managed by the container

Stateful session bean



Annotations (@)

- Adding metadata to classes, methods, variables:
- calling services, validation, dependency injection, accessing resources (for example datasources)
- @Override (checks if the method really overrides its parent)
- Evolution: configs in .xml files –> annotation

Annotations and EJBs

POJO (Plain Old Java Object) + Annotion –> EJB

```
@Local
public interface PersonServiceLocal{
}

@Stateless
public class PersonService implements PersonServiceLocal{

   @PostConstruct
    public void onInitializes(){
      //a Service létrejötte után fut le
    }
}
```

Annotations and lifecycle callbacks

- Bean methods can be bound to lifecycle steps through annotations:
- @PostConstruct – runs after bean instance is created
- @PreDestroy – runs before bean instance is destroyed
- @PrePassivate, @PostActivate

Obtaining reference on EJB

- Cannot be instantiated with new, because:
  - we do not know its class only the interface we implement
  - we want to fully use the capabilities of the container (do we need a new instance or is there an existing one we could use)
- Ways for getting a reference:
- Injection through annotation at the place of intended usage in the code (@Inject or @EJB)
- naming service (JNDI lookup)

JNDI

- Java Naming and Directory Interface: *Java API: enables the Java client to obtain reference on beans through looking it up by name*
- *phone registry for beans*
- Each EJB component and resource receives a JNDI name after deployment – which is maintained by the JNDI provider

java:global[/<app-name>]/<module-name>/<Bean-name>[!<fully-qualified-interface-name>]

JNDI example

- PersonService JNDI (you can view the JNDI names in the server boot log):
- java:global/gyakorlat_2/PersonService!test.view.PersonService
- java:app/gyakorlat_2/PersonService!test.view.PersonService
- java:module/PersonService!test.view.PersonService
- java:global/gyakorlat_2/PersonService
- java:app/gyakorlat_2/PersonService
- java:module/PersonService
- DataSource elérése:
- java:/laborDS

EJB referencing

Injection

```
@Named
public class PersonBean{

      @EJB
       private PersonServiceLocal personService;
}
```

```
@Named
public class PersonBeans{

      @Inject
       private PersonServiceLocal personService;
}
```

EJB referencing

JNDI lookup (in non-managed context)

```
public class PersonBean{

    public void lookup(){

        try {
            // Create the initial context
           Context ctx = new InitialContext(env);
          // Look up an object
            Object obj =
            ctx.lookup(”java:global/gyakorlat_2/PersonService”);
         //castolás …
       } catch (NamingException e) {
            System.err.println("Problem looking up " + name + ": " + e);
       }
   }
```

Singleton(@Singleton)

- Container guarantuees that only one instance will exist in each JVM of the beans with @Singleton annotation
- Container can be forced to create the instance when the software is deployed (@Startup)
- Hierarchy between Singletons can be defined(@DependsOn)
- @ConcurrencyManagement(Container vs Service)
- Threads & concurrency: @Lock(READ) or @Lock(WRITE)