

**Middleware GSD (WS 2017)**

**Question and Answer**

**on**

**Exercise 04**

**Prof. Dr. Rainer Todtenhöfer**

**Submitted By**

**Mahbubur Rahman**

**Matrikel Nr: 250154**

**M.Sc in Global Software Development**

**Hochschule Fulda University of Applied Sciences**

**Submission Date**

**22-11-2017**

**Please summarize pages 23-1 to 23-8 (Dependency Injection) and 32-1 to 32-14 (EJB Basics) of the Java EE 7 tutorial on about one page each!**

Context Dependency Injection:

Context Dependency Injection is the one of the popular mechanism in Java EE where any kind of Java class can be automatically instantiated and injected into dependent class before its instantiation.

The main benefits is the class does not need to care about its dependencies either they are instantiated or not because these are automatically done by CDI. At earlier, a class is responsible for its dependencies initialization using init method by creating objects or passing as parameters.

Dependency Injection maintains a life cycle for injectable objects. So CDI updates the bean definition of Java beans and Java EE beans. The life cycle duration of a CDI bean is measured based on scopes. There are few types of scopes in CDI. The main 3 scopes are given bellow:

1. Request Scope:  
     
   The annotation of request scope is @RequestScoped. If a CDI bean is annotated with request scope, then the bean object life cycle is limited within single HTTP request that means when an HTTP request is sent to server, the server instantiate the class and inject it into another class and after response is sent back to client, the bean object dies and it is deleted by garbage collector.
2. Session Scope:  
     
   The annotation of session scope is @SessionScoped. Within this scope, a bean's lifecycle duration is more than request scope and it can deal with multiple HTTP requests of a single user.
3. Application Scope:   
     
   The annotation of application scope is @ApplicationScoped. This scope has long lifecycle that means it starts when application is started and ends when application will be shut down. It stays with application and shares itself throughout the application so that multiple user with multiple HTTP requests shared this CDI bean.

To apply dependency injection, we need to annotate the attribute of CDI bean interface with @Inject annotation. Then in runtime, the container look for the implementation class of injectable interface. After that it creates object and injects into the another class. The object is maintained as per lifecycle context model that is described in CDI specification.

If multiple implementations are found of a single interface, then it is resolved by another annonation named @Qualifier.

To define a qualifier, we need another two annotations named @Retention(RUNTIME) and @Target({METHOD, TYPE, PARAMETER, FIELD})

For Example:

@Qualifier

@Retention(RUNTIME)

@Target({METHOD, FIELD, TYPE, PARAMETER})

public @interface Informal {}

@RequestScoped

@Default

public class A {

int a;

public int getNumber() {

return a;  
 }

}

@RequestScoped

@Informal

public class B extends A {

int b;

@Override

public int getNumber() {

return b;  
 }

}

public class C {

@Inject

A a1; // it will inject the object of class A

@Inject

@Informal

A a2; // it will inject the object of class B

}

EJB Basics:

Enterprise Java Bean(EJB) is a server side component that wraps application's business logics inside it. It is focused on only business rules of a application. So the benefit of using EJB is developer can give concentration on more and complex business logics implementation rather than system configuration tasks. The system configuration tasks like security, authorization, resource management are done by EJB containers. Using EJB, it is easy to develop scalable and distributed application because ejb modules can be deployed in different machines and its transparent to clients. So to design a Java EE application, at first we need to think how we give access our enterprise beans to clients. There are three ways, one is local access, another two are remote access and web services respectively. By default remote access is the better choice from future point of view because if we want to make growing system then its the good choice and we can make our application distributed and can run from different locations.

There are two types of EJB. They are described bellow:

1. Session Bean:  
     
   Session beans are responsible for doing client's tasks. Within session bean, all business logics are implemented. There are three types of session bean. They are:  
   1. Stateful Session Bean:  
        
      Stateful session bean maintains state throughout the client server interaction. There are 3 stages in its lifecycle. They are “Does not Exists”, “Ready” and “Passive”. The values of instance variables are the state of a bean and it continues until user ends the session.
   2. Stateless Session Bean:

Stateless session bean has no states. When user invokes a method then the session starts and after invoking method the session closes. As a result, this type of session bean can handle multiple users interactions within short duration of time. In its lifecycle, there are 2 stages. They are “Does not Exists” and “Ready”.

* 1. Singleton Session Bean:

Singleton session bean is more like stateless session bean but the difference is it stays throughout application life time and create once at a time during application startup. There is also a limitation in number of singleton session bean. One application can contains only one singleton session bean. In its lifecycle it has 2 stages like stateless session bean.

Message Driven Bean:

Message driven bean is used as listener of java messaging api that means it receives messages instead of events. In its lifecycle, there are 2 stages. They are “Does not Exists” and “Ready”.