INTRO

where we are now: **EJB3.2** and **CDI 1.1** as of JavaEE7

- technically its CDI & EJB & Managed Beans & Interceptors
- concepts are very similar to any other IoC and DI designs

CDI

- defines lifecycle bound contexts
- typesafe dependency injection
- event notifications
- bean decorations - AOP
- SPI for integration with javaee containers

TERMINOLOGY

- injectable object(managed beans, EJBs, java ee res) - bean
- instance of bean - contextual instances
- runtime environment - container
- bean-level metadata - annotations
- application level metadata - xml descriptor (beans.xml)

BEAN

A Java EE component managed by container according to the lifecycle context

- can have a name(EL identifier) `@Named`
- needs an empty constructor or one with `@Inject`

QUALIFIERS

Discriminator for multiple beans implementing the same type.

```
@Synchronous
class SynchronousPaymentProcessor implements PaymentProcessor {
   ...
}
@Asynchronous
class AsynchronousPaymentProcessor implements PaymentProcessor {
   ...
}
@Qualifier
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface Synchronous {}
...
@Inject @Synchronous PaymentProcessor processor;
```

ALTERNATIVES & STEREOTYPES

Alternatives

- @Alternative on the Bean or producer
- you have to explicitly tell which of the alternatives you want

Stereotype

- bundle common behaviour into a single responsible
- @Stereotype

```
@RequestScoped
@Secure
@Transactional
@Stereotype
@Target(TYPE)
@Retention(RUNTIME)
public @interface Action {}
@Action
public class SaveOrderAction {}
```

SCOPES

- @RequestScoped
- @SessionScoped
- @ApplicationScoped
- @ConversationScoped
- @Dependent
- Custom(e.g. ProcessScoped, ...)

PRODUCERS

- source of objects to be injected
- can provide beans, non-beans, can add specific features, details

```
public class MagicFactory {
    @Produces
    @PersistenceContext
    private EntityManager em; //producer field

    //producer method - has access to metadata of the point where we want injection
    @Produces Logger getLogger(InjectionPoint injectionPoint) {
        return Logger.getLogger(injectionPoint.getMember().getDeclaringClass().getSimpleName());
    }
}
```

CONTAINER PROVIDED BEANS

- `javax.transaction.UserTransaction`
- `javax.security.Principal`

Within servlet container

- `javax.servlet.http.HttpServletRequest`
- `javax.servlet.http.HttpSession`
- `javax.servlet.ServletContext`

INTERCEPTORS

AOP for java, intercept @AroundInvoke, @AroundTimeout,...

```
<beans ...>
    <interceptors><!--order is important-->
        <class>com.acme.myfwk.LoggingInterceptor</class>
    </interceptors>
</beans>
```

```
@Interceptor @Logging//interceptor and some qualifier
public class LoggingInterceptor {
  //. . .
}
@Logging
//or explicitly @Interceptors(LoggingInterceptor.class)
public class Bean{}
```

COMMON PROBLEMS

- unsatisfied dependency - there is no bean that can be injected, maybe a misconfiguration, missing annotation, missing beans.xml, incorrect deployment
- ambiguous dependency - multiple beans are eligible - maybe hierarchy issue, incorrect deployment, @alternatives...
- beans.xml META-INF/beans.xml , or, in a war, WEB-INF/beans.xml or WEB-INF/classes/META-INF/beans.xml

DOWN THE RABBIT HOLE

- extensions
- decorators
- events
- bean discovery

EJB3

EJB >= CDI

server-side component that encapsulates the business logic, integrated transaction handling

- session beans (stateless, stateful,singleton)
- message driven beans

BENEFITS

- transaction handling
- startup hook, clustering
- timers
- Web services
- multi threading
- +/- pool - throttling

ACCESS & IMPLEMENTATION

- local(think in-VM) @Local
- remote(think remote network) @Remote
- no interface local

Must be annotated with `@Stateless`, `@Statefull`, `@Singleton`, `@MessageDriven`

TRANSACTION MGMT

- container by default
- `@TransactionAttribute(REQUIRED,REQUIRES_NEW,NOT_SUPPORTED)`
- transaction rollback either with `Context.setRollbackOnly()` or your exception needs `@ApplicationException(rollback=true)`

HANDS-ON

Import `hands-on/SampleProjectCDI` and play around. Try removing qualifiers and adding more beans.

KNOWLEDGE CHECK

- what is a qualifier, what can it be used for?
- what can be injected?
- what is a CDI producer?
- what is an interceptor?
- how do I add an interceptor to some bean?
- what does this do: `@Inject @Polite GreetingService service` ?
- can you inject `@RequestScoped` bean using `@EJB`? Why?
- what are the main differences between EJB and CDI beans? When would you use them?
- write a simplest possible tracing interceptor that can be disabled for specific method e.g.

```
@Stateless
@Trace
public class MyService {
    public void sayHi() {} //should be traced by interceptor

    public void sayHello() {} //should be traced by interceptor

    @Trace(false)
    public void sayWassup() {} // please don't trace this gibberish
}
```

HOMEWORK

- get all sample projects running
- try debug mode - arquillian managed server debugging
- import cdi/events demo project from github repo
- (optional) go over other examples in CDI examples in the same repo
- import ejb projects from github repo have a look at async-ejb
- learn/refresh basic http (for example here)
- google around for REST api best practices (try this if you are lazy)
- have a look at some of the popular REST apis github, eveentbrite,....