



Inventory #maven

Interceptor, JNDI variable, CDI, JSON Binding

Óbudai Egyetem, Java Enterprise Edition

Műszaki Informatika szak

Labor 9

Bedők Dávid
2018-03-14
v1.0



Feladat: Egy egyszerű **Inventory** alkalmazás létrehozása, melyben az egyes tételeknek (`InventoryItem`) az alábbi adatát tároljuk memóriában: *referencia* (`String`), *megnevezés* (`String`), *típus* (felsorolás érték: `BOOK`, `DISK`, `CASSETTE` vagy `MAGAZINE`) és *érték* (`int`).

A feladat célja az alábbi, korábban nem érintett területek bemutatása:

- ▷ JavaEE **Interceptor**
- ▷ **CDI**
 - Interceptor CDI segítségével
 - **Qualifier** alapú injection
 - Dinamikus injection
 - **Eseménykezelés**
 - Injection factory segítségével
 - Direkt injection (programozott CDI bean kérése)
- ▷ JNDI-on keresztüli alkalmazás konfiguráció
- ▷ **JSON Binding** annotációk



```
[gradle|maven]\jbossinventory
```



- ▷ **Adat réteg** (inv-persistence)
 - *Singleton Session Bean* segítségével, memóriában tárolja az előre definiált *inventory* elemeket.
 - `InventoryItem` entitás, `InventoryType` enum
- ▷ **Logikai réteg** (inv-ejbservice)
 - Elsősorban ebben a rétegben jelennek majd meg azok a technikák, melyeket még nem érintettünk eleddig.
 - `InventoryItemBean` DTO
 - az `InventoryType` helyett `String`-ként tárolja az *inventory* típusát
 - `InventoryItemBeanConverter` alakít a két típus között
 - `InternationalInventoryItemBean` DTO
 - az `InventoryItem` egy speciális üzleti célra alkotott nézete
 - `InternationalInventoryItemConverter` alakít a két típus között
- ▷ **Megjelenítési réteg** (inv-webservice)
 - RESTful webszolgáltatásokat tartalmaz a bemutatott technikák bemutatásának céljából
 - Context root: **inventory**
 - RESTful API application path: **api**



Az *inventory* elemeket memóriában tárolja az InventoryHolder SSB. A bean inicializálja az adatokat, illetve képes visszaadni azokat olvasásra. Két lekérdező SLSB lett definiálva a tárolási rétegen belül:

▷ InventoryFinder

```
1 @Local
2 public interface InventoryFinder {
3     InventoryItem get(String reference);
4     List<InventoryItem> list(InventoryType type);
5 }
```

▷ InventorySearch

```
1 @Local
2 public interface InventorySearch {
3     List<InventoryItem> list(InventoryType type, String nameTerm);
4 }
```

Az InventorySearch implementációját opcionálisan **Groovy** nyelven készítjük majd el, ezért van ketté szedve a szolgáltatás.



RESTful webszolgáltatások segítségével fogjuk megszólítani az *Inventory* rendszert.

BasicRestService

- ▷ <http://localhost:8080/inventory/api/basic/{reference}>
 - Megadott azonosítóval rendelkező *inventory* elem visszaadása JSON formátumban.
- ▷ <http://localhost:8080/inventory/api/basic/list/{type}>
 - Megadott típusnak (*InventoryType enum*) megfelelő *inventory* elemek visszaadása JSON formátumban.
- ▷ <http://localhost:8080/inventory/api/basic/list/{type}/{nameTerm}>
 - Megadott típusnak és név töredéknek (kezdetnek) megfelelő *inventory* elemek visszaadása JSON formátumban.

A réteg további webszolgáltatásokat is tartalmazni fog. A használatot megelőzően visszatérünk majd ezek üzleti képességeire is.



Részfeladat: A `BasicRestService` RESTful webservice szolgáltatás metódusainak eredményét egészítjük ki *copyright* információval (© [YEAR] Inventory) automatikusan.

Eredeti JSON

```
1 {
2   "reference": "LOR78",
3   "name": "Lorem20",
4   "type": "BOOK",
5   "value": 78
6 }
```

Módosított JSON

```
1 {
2   "reference": "LOR78",
3   "name": "Lorem20 @ 2018 Inventory",
4   "type": "BOOK",
5   "value": 78
6 }
```

A részfeladat könnyedén implementálható néhány új kódsor beillesztésével, azonban ez a megvalósítás több szempontból is előnytelen lenne:

- ▷ Ismétlődő, redundáns és *boilerplate* kódot eredményez (ismétlődő hívások számos üzleti metódusban megjelennek).
- ▷ Egymáshoz nem kapcsolódó üzleti felelőségek egy üzleti metódusban helyezkednének el (OO *encapsulation* elvét sérti)
- ▷ Ha igény van ezen kiegészítő felelősség ki- illetve bekapcsolására, akkor ezt csak újabb komplexitás bevezetésével tudnánk eszközölni

Mind ezek miatt a feladatot JavaEE **interceptor** segítségével oldjuk meg!

Interceptor

Aspect-Oriented Programming (AOP)

- ▷ JSR 318, Enterprise JavaBeans 3.1
- ▷ Eljárásokkal összefüggésben illetve életciklus elemek körül definiálhatóak.
- ▷ **Létrehozás**: Osztályként vagy metódusként is implementálhatóak (mi csak osztályként valósítjuk meg a *Separation of Concern* végett¹).
- ▷ **Kötés**: A felhasználás helyén konfigurálható a `javax.interceptor.Interceptors` (s!) annotációval, XML konfigurációval illetve CDI segítségével is. Javasolt az utóbbi alkalmazása, de bemutatásra kerül a többi megoldás is.

Interceptor Metadata Annotation	Leírás
<code>javax.interceptor.AroundInvoke</code>	A metódus megjelölése interceptor-ként.
<code>javax.interceptor.AroundTimeout</code>	Enterprise bean timeout esetén a metódust timeout interceptor-ként jelöli meg.
<code>javax.annotation.PostConstruct</code>	A bean létrehozás életciklus interceptora.
<code>javax.annotation.PreDestroy</code>	A bean megszüntetés életciklis interceptora.

¹ https://en.wikipedia.org/wiki/Separation_of_concerns

Copyright interceptor

Az *interceptor*-ok részei az *EJB context*-nek, alapesetben nem szükséges "regisztrálni" őket, a kötés során automatikusan aktiválódnak.

```
1  [...]
2  public class CopyrightInterceptor implements Serializable {
3
4      @AroundInvoke
5      public Object decorateCoinWithCopyright(InvocationContext context) throws Exception {
6          int currentYear = Calendar.getInstance().get(Calendar.YEAR);
7          final Object result = context.proceed();
8          if (result instanceof InventoryItemBean) {
9              this.modifyInventoryItemName(result, currentYear);
10         } else if (result instanceof List) {
11             List<?> resultList = (List<?>) result;
12             for (Object resultItem : resultList) {
13                 if (resultItem instanceof InventoryItemBean) {
14                     this.modifyInventoryItemName(resultItem, currentYear);
15                 }
16             }
17         }
18         return result;
19     }
20
21     private void modifyInventoryItemName(Object object, int currentYear) {
22         InventoryItemBean item = (InventoryItemBean) object;
23         String description = item.getName();
24         item.setName(description + " @ " + currentYear + " Inventory");
25     }
26 }
```

Bejárhatjuk a paramétereket, azok értékeit, típusait. Saját annotációk használata esetén *Reflection API*-val további vezérlést is beépíthetünk!

Egy *service* hívás köré szeretnénk ezt beágyazni (*@AroundInvoke* annotáció). A valós hívást mi kezdeményezzük (*context.proceed()*), és ennek megfelelően akár meg is akadályozhatjuk, vagy módosíthatjuk a kimenetét mielőtt visszatérünk.

CopyrightInterceptor.java

Interceptor bekötése

ejb-jar.xml segítségével

XML alapú konfiguráció

Nem foglalkoztunk vele korábban, de minden annotációval vezérelt beállítás megadható egy EJB *module* esetén az `ejb-jar.xml` segítségével is.

Együttes jelenlét esetén mindig az XML konfiguráció az erősebb. Az XML konfiguráció bár kényelmetlenebb, nagy előnye hogy nem visz be új *compile time* függőségeket (pl. nem kell az alkalmazott annotációk miatt API *jar*-okat a *classpath*-ra tenni).

```
1 <interceptor-binding>
2   <target-name>
3     hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl
4   </target-name>
5   <interceptor-class>
6     hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor
7   </interceptor-class>
8   <method-name>getInventory</method-name>
9 </interceptor-binding>
```

Interceptor bekötése

@Interceptors annotáció segítségével

Ezek helyett a CDI binding egy még kényelmesebb megoldás lesz, a CDI megismerése után azt fogjuk alkalmazni!

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4     @EJB
5     private InventoryFinder finder;
6
7     @Inject
8     private InventoryItemBeanConverter converter;
9
10    @Override
11    @Interceptors({ CopyrightInterceptor.class })
12    public InventoryItemBean getInventoryItem(String reference) throws AdaptorException {
13        return this.converter.to(this.finder.get(reference));
14    }
15
16    @Override
17    @Interceptors({ CopyrightInterceptor.class })
18    public List<InventoryItemBean> getInventoryItems(final InventoryType type) throws
        AdaptorException {
19        return this.converter.to(this.finder.list(type));
20    }
21    [...]
22 }
```

Az `InventoryItemBean` konverter injectálására a CDI bevezetője után visszatérünk! A korábban megismert SLSB segítségével is implementálható.

Több *interceptor* osztály felsorolható az @Interceptors annotáció segítségével.

InventoryFacadeImpl.java



<http://localhost:8080/inventory/api/basic/LOR78>

```
13:28:43,630 INFO [hu.qwaevisz.inventory.webservice.SearchRestServiceBean] (HTTP-1)
Get Inventory by LOR78 (reference)
13:28:43,636 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-1) Copyright Interceptor activated in getInventory
13:28:43,674 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-1) Add copyright for LOR78's name.
```

```
1 {
2     "reference": "LOR78",
3     "name": "Lorem20 @ 2018 Inventory",
4     "type": "BOOK",
5     "value": 78
6 }
```

Továbbfejlesztési lehetőségek

Interceptorok és CDI kapcsolata

Az *interceptor*-ok használata egyszerű és nagyszerű, azonban még elegánsabbá tehetjük CDI segítségével.

- ▷ Az `@Interceptor` (egyes szám) annotáció segítségével megjelöljük az *interceptor* osztályt.
- ▷ Létrehozunk egy új annotációt, melyet a kötéshez fogunk használni. Ezt az új annotációt ellátjuk az `@InterceptorBinding` annotációval, mely annotációk annotálására használható.
- ▷ Az új annotációval dekoráljuk azokat a helyeket, ahol szeretnénk érvényesíteni az *interceptor*-t.

Az így létrehozott *interceptor* kötések **alapértelmezetten kikapcsoltak**, a `beans.xml`-ben aktiválni szükséges őket. A `beans.xml` a CDI *deployment descriptor*-a, jelenléte aktiválja a CDI lehetőségeket (akár üres állomány is lehet).

- ▷ JSR 299, JSR 330, JSR 316
- ▷ *Expression Language* (EL) integráció *Java Server Faces* illetve JSP lapokban való komponens eléréshez
- ▷ Osztályok, injektált elemek dekorációja
- ▷ **Interceptor**-ok regisztrálása és kötése "elegánsabb", "egyszerűbb" módon
- ▷ **Eseménykezelés** bean-ek között
- ▷ *Web conversion scope*, mely kiegészíti a *request*, *session* és *application* *Java Servlet* specifikációt
- ▷ *Service Provider Interface* (SPI), mely lehetővé teszi 3rd party keretrendszerek JavaEE 6 környezetbe ágyazását
- ▷ A CDI használatával a `String` kulcs alapján működő erőforrás lekérést **type-safe** módon tudjuk kezelni (erre elsősorban ott van szükség, ahol több implementáció létezik egy adott szolgáltatáshoz)

- ▷ A @EJB annotáció használata helyett a CDI @Inject annotációját is használhatjuk a továbbiakban.
 - Az @EJB annotációt csak ott lehet használni, ahol EJB *context* elérhető (pl. egy sima POJO-ban nem, de egy *servlet*-ben, *session bean*-ben igen, vagy bármely osztályban ami az EJB *context* látókörébe kerül valamilyen módon).
 - A CDI használható szinte az összes Java osztályban (mely a CDI *context* látókörébe kerül), *session bean*-ekben, stb. (de nem használható: *Message Driven Bean*-ekben és *Remote EJB*-kben).
- ▷ *Classpath*-ra egy `beans.xml` elhelyezése szükséges azokban az EJB *module*-okban, ahol használni szeretnénk (`src/main/resources/beans.xml`). A használat elkezdéséhez elegendő egy teljesen üres file is (<http://www.cdi-spec.org/faq/>).

InterceptorItemBean konverter

```
1 public interface InventoryItemBeanConverter {
2     InventoryItemBean to(InventoryItem item);
3     List<InventoryItemBean> to(List<InventoryItem> items);
4 }
```

Nincs az interface-en
@Local annotáció!

InventoryItemBeanConverter.java

```
1 public class InventoryItemBeanConverterImpl implements InventoryItemBeanConverter {
2
3     @Override
4     public InventoryItemBean to(InventoryItem item) { [...] }
5
6     @Override
7     public List<InventoryItemBean> to(List<InventoryItem> items) {
8         return items.stream().map(this::to).collect(Collectors.toList());
9     }
10 }
```

Nincs az osztályon
@Stateless annotáció!

InventoryItemBeanConverterImpl.java

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryItemBeanConverter {
3
4     @Inject
5     private InventoryItemBeanConverter converter;
6
7     [...]
8 }
```

Az @EJB helyett a @Inject annotációt használjuk. A háttérben ugyanúgy egy proxy-t kapunk, a CDI megtalálja hogy csak egy implementáció van, mely automatikusan a @Default *qualifier*-rel fog rendelkezni.

InventoryItemBeanConverter.java

CDI vs. EJB

EJB	CDI
<i>Container</i> szolgáltatásokat biztosít (tranzakció kezeléssel, <i>security</i> -val, konkurencia kezeléssel, stb.)	Mudularizációt, <i>separation of concerns</i> -t, eseménykezelést, <i>injection</i> -t biztosít.
Az EJB-k CDI bean-ek is, ennek minden előnyével.	-
@Stateful, @Stateless, @Singleton felbontás	@RequestScoped, @SessionScoped, @ApplicationScoped, @ConversationScoped (JSF) vagy custom <i>scope</i> felbontás
@Startup annotáció segítségével előre inicializálható	nincs CDI megfelelője
@Asynchronous metódus hívás	nincs CDI megfelelője
@Schedule annotációval ütemezett indítás	nincs CDI megfelelője
@TransactionAttribute annotáció	Nem teljesen <i>container managed</i> tranzakcióról beszélhetünk, de CDI bean-ek esetén is megoldott
@Singleton esetén egyszerű szinkronizáció támogatott a @Lock(READ) és @Lock(WRITE) annotációk segítségével	nincs CDI megfelelője

Nincs teljesítménykülönbség a gyakorlatban a két megközelítés között (azonos célú és igényű felhasználás esetén). A CDI példány feloldása egy kicsit komplexebb, de ez nem jelenik meg a teljesítményben.

Azonos tervezési minta (*proxy pattern*).



Részfeladat: A `BasicRestService` RESTful webszolgáltatás metódusainak hívásait (hívott paraméterekkel együtt) automatikusan naplózzuk, és mérjük a szolgáltatás hívások futási idejét.

A feladatot JavaEE **interceptor** segítségével oldjuk meg oly módon, hogy az *interceptor* regisztrációját CDI segítségével eszközöljük.

Interceptorok regisztrációja CDI segítségével

A `javax.interceptor.Interceptor` annotáció segítségével regisztráljuk az osztályt a CDI számára.

```
1 [...]
2 @Interceptor
3 public class LoggedInterceptor implements Serializable {
4     [...]
5 }
```

LoggedInterceptor.java

Az így létrehozott interceptor alapértelmezésben **inaktív** lesz, aktiválni a `beans.xml`-ben lehet.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5         http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
6     <interceptors>
7         <class>hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor</class>
8     </interceptors>
9
10 </beans>
```

inv-ejbservice | src | main | resources | beans.xml

Naplózás és futási idő mérése

```
1 @Interceptor
2 public class LoggedInterceptor implements Serializable {
3
4     private static final Logger LOGGER = Logger.getLogger(LoggedInterceptor.class);
5
6     @AroundInvoke
7     public Object logMethodInvocations(InvocationContext context) throws Exception {
8         final StringBuilder info = new StringBuilder();
9         info.append(context.getTarget().getClass().getName()).append(".").append(context.getMethodName());
10        final Object[] parameters = context.getParameters();
11        if (parameters != null) {
12            int i = parameters.length - 1;
13            for (final Object parameter : parameters) {
14                info.append(parameter.toString());
15                if (i > 0) {
16                    info.append(",");
17                }
18                i--;
19            }
20        }
21        info.append(")");
22        LOGGER.info("Entering: " + info.toString());
23        final long start = System.currentTimeMillis();
24        final Object result = context.proceed();
25        final long end = System.currentTimeMillis();
26        LOGGER.info("Exiting: " + info.toString() + " - running time: " + (end - start) + "
27            millisecond(s)");
28        return result;
29    }
}
```

Logged

@InterceptorBinding annotáció

```
1 package hu.qwaevisz.inventory.ejbservice.interceptor;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Inherited;
5 import java.lang.annotation.Retention;
6 import java.lang.annotation.RetentionPolicy;
7 import java.lang.annotation.Target;
8
9 import javax.interceptor.InterceptorBinding;
10
11 @Inherited
12 @InterceptorBinding
13 @Retention(RetentionPolicy.RUNTIME)
14 @Target({ ElementType.TYPE, ElementType.METHOD })
15 public @interface Logged {
16
17 }
```

Az @InterceptorBinding annotáció *interceptor*-ok bekötésére használható annotáció, vagyis **annotációk annotálására használható**. Az így létrejött annotációt (@Logged) egyrészt az *interceptor* osztályán, másrészt a felhasználási helyeken fogjuk alkalmazni.

Logged.java

<< annotation >>

```
@InterceptorBinding  
Logged
```

Utolsó lépésként a most létrehozott @Logged annotációval el kell látnunk mind az *interceptor* osztályát, mind az összes kötési helyet.

<< interceptor class >>

```
@Logged  
@Interceptor  
LoggedInterceptor
```

<< interceptor method >>

```
@AroundInvoke  
logMethodInvocations(..)
```

<< session bean class >>

```
@Stateless  
InventoryFacadeImpl
```

<< business method >>

```
@Logged  
getInventoryItems(..)
```

Interceptor osztályának módosítása

```
1  [...]
2  @Logged
3  @Interceptor
4  public class LoggedInterceptor implements Serializable {
5
6      [...]
7
8      @AroundInvoke
9      public Object logMethodInvocations(InvocationContext context)
10         throws Exception {
11         [...]
12     }
13 }
```

LoggedInterceptor.java

Interceptor bekötése CDI segítségével

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4     @EJB
5     private InventoryFinder finder;
6
7     @Inject
8     private InventoryItemBeanConverter converter;
9
10    @Logged
11    @Override
12    @Interceptors({ CopyrightInterceptor.class })
13    public List<InventoryItemBean> getInventoryItems(final InventoryType type) throws
        AdaptorException {
14        return this.converter.to(this.finder.list(type));
15    }
16
17    @Inject
18    private InventorySearch search;
19
20    @Logged
21    @Override
22    public List<InventoryItemBean> getInventoryItems(final InventoryType type, final
        String nameTerm) throws AdaptorException {
23        return this.converter.to(this.search.list(type, nameTerm));
24    }
25
26    [...]
27 }
```

A `@Logged` annotációval minden olyan metódust megjelölhetünk, melynél szeretnénk a hívás elején és végén naplózni, illetve a kettő között eltelt futási időt mérni.



<http://localhost:8080/inventory/api/basic/list/BOOK>

```
13:29:52,659 INFO [hu.qwaevisz.inventory.webservice.SearchRestServiceBean] (HTTP-3)
Get Inventories by BOOK type
13:29:52,659 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-3) Copyright
Interceptor activated in getInventories
13:29:52,660 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(HTTP-3) Entering:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK)
13:29:52,672 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(HTTP-3) Exiting:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK) -
running time: 12 millisecond(s)
13:29:52,672 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-3) Add copyright for LOR42's name.
13:29:52,672 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-3) Add copyright for LOR78's name.
13:29:52,672 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-3) Add copyright for LOR34's name.
13:29:52,672 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.CopyrightInterceptor]
(HTTP-3) Add copyright for SIT78's name.
```


Naplózás és futási idő tesztelése - II

RESTful webszolgáltatáson keresztül



<http://localhost:8080/inventory/api/basic/list/BOOK/Lorem>

```
13:31:18,760 INFO [hu.qwaevisz.inventory.webservice.SearchRestServiceBean] (HTTP-2)
Get Inventories by BOOK type and Lorem name
13:31:18,761 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(HTTP-2) Entering:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK,Lorem)
13:31:19,168 INFO [hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(HTTP-2) Exiting:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK,Lorem)
- running time: 407 millisecond(s)
```

```
1 [
2   {
3     "reference": "LOR42",
4     "name": "Lorem10 @ 2018 Inventory",
5     "type": "BOOK",
6     "value": 42
7   },
8   {
9     [...]
10  },
11  [...]
12 ]
```



Részfeladat: Az *inventory* elemeket RESTful webszolgáltatásokon keresztül lehet megvásárolni a rendszerből. Azonban időszakosan nem a normál, hanem egy kedvezményes (20%-al csökkentett) áron lehet a termékeket megszerezni. A termék ára természetesen nem változik meg az adat rétegben, csupán a visszaadott `InventoryItemBean` ára lesz különböző.

Külön *endpoint*-okon legyen lekérdezhető a normál és a kedvezményes árú `InventoryItemBean` is, függetlenül az akciós időszaktól.

Az ármódosítást egy `CostService` határozza meg, mely egy mezei *interface* legyen, egy `NormalCostService` és egy `PromotionalCostService` implementációval. Promóciós időszakban CDI *qualifier*-ek segítségével *inject*-áljuk a megfelelő implementációt.

Akciós időszaktól független lekérdezés során **dinamikusan** *inject*-áljuk a megfelelő implementációt!



PurchaseRestService

- ▷ <http://localhost:8080/inventory/api/purchase/{reference}>
 - Megadott azonosítóval rendelkező *inventory* elem megvásárlása az időszak függvényében.
- ▷ <http://localhost:8080/inventory/api/purchase/{reference}/{pricing}>
 - Megadott árazási stratégiának függvényében az *inventory* elem megvásárlása.
 - PricingStrategy (enum): STANDARD vagy DISCOUNT

Qualifiers

JavaEE CDI vonatkozásában

A **qualifier** egy annotáció, mely *bean*-ek annotálására használható. Egy annotáció attól lesz *qualifier*, hogy rendelkezik a `javax.inject.Qualifier` annotációval.

A *qualifier*-ek segítségével *type-safe* módon lehet használni pl. egy EJB két különféle implementációját (a kért *qualifier*-rel rendelkezőt lehet *inject*-álni), vagy mindehhez nem is kellene *session bean*-ek (*CDI bean*-ek környezetében értelmezzük).

Ha egy *bean*-nek nincs *qualifier*-e, akkor automatikusan kap egy `@Default` *qualifier*-t.

Több implementáció az EJB világában

Ha egy EJB-nek több implementációja van a *bean* nevét jelezni kell az annotációk használata során (pl. `@Stateless(name="lorem")` és `@EJB(beanName="lorem")`). Ez teszi ezt a megoldást nem *type-safe* módszerré (természetesen a `String` értékek helyet lehet egy konstanst használni, szokták ezt a `@Local/@Remote` interface-ben elhelyezni).

Promóciós qualifier-ek

```
1 package hu.qwaevisz.inventory.ejb.service.qualifier;  
2 [..]  
3 @Qualifier  
4 @Retention(RetentionPolicy.RUNTIME)  
5 @Target({ ElementType.TYPE, ElementType.METHOD,  
6           ElementType.FIELD, ElementType.PARAMETER })  
7 public @interface Standard {  
8 }
```

Standard.java

```
1 package hu.qwaevisz.inventory.ejb.service.qualifier;  
2 [..]  
3 @Qualifier  
4 @Retention(RetentionPolicy.RUNTIME)  
5 @Target({ ElementType.TYPE, ElementType.METHOD,  
6           ElementType.FIELD, ElementType.PARAMETER })  
7 public @interface Discount {  
8 }
```

Discount.java

CostService és implementációi

```
1 package hu.qwaevisz.inventory.ejb.service.cost;
2 public interface CostService {
3     int getCost(int originalValue);
4 }
```

CostService.java

```
1 [...]
2 @Standard
3 public class NormalCostService implements CostService {
4     @Override
5     public int getCost(int originalValue) {
6         return originalValue;
7     }
8 }
```

NormalCostService.java

```
1 [...]
2 @Discount
3 public class PromotionalCostService implements CostService {
4     private static final float DISCOUNT_PERCENT = 20;
5     @Override
6     public int getCost(int originalValue) {
7         return Math.round(originalValue * (1 - (DISCOUNT_PERCENT / 100)));
8     }
9 }
```

PromotionalCostService.java

Qualifier alapú *injection*

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4     @EJB
5     private InventoryFinder finder;
6
7     @Inject
8     private InventoryItemBeanConverter converter;
9
10    @Inject
11    @Discount
12    private CostService costService;
13
14    @Logged
15    @Override
16    public InventoryItemBean buyInventoryItem(String reference) throws AdaptorException {
17        final InventoryItemBean bean = this.converter.to(this.finder.get(reference));
18        bean.setValue(this.costService.getCost(bean.getValue()));
19        return bean;
20    }
21 }
```

InventoryFacadeImpl.java

Promóciós ár tesztelése

RESTful webszolgáltatáson keresztül



<http://localhost:8080/inventory/api/purchase/LOR78>

```
1 {  
2   "reference": "LOR78",  
3   "name": "Lorem20",  
4   "type": "BOOK",  
5   "value": 62  
6 }
```

A termék ára **78** helyett akciós időszakban csupán **62 (EUR)**, mivel $78 * 0.8 = 62.4$.

Dinamikus injection

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3     [...]
4     @Any
5     @Inject
6     private Instance<CostService> dynamicCostService;
7
8     @Logged
9     @Override
10    public InventoryItemBean buyInventoryItem(String reference, PricingStrategy pricing)
11        throws AdaptorException {
12        final InventoryItemBean bean = this.converter.to(this.finder.get(reference));
13        bean.setValue(this.getCostService(pricing).getCost(bean.getValue()));
14        return bean;
15    }
16    private CostService getCostService(PricingStrategy pricing) {
17        CostService service = null;
18        switch (pricing) {
19            case STANDARD:
20                service = this.dynamicCostService.select(new AnnotationLiteral<Standard>() {
21                    private static final long serialVersionUID = 1L;
22                }).get();
23                break;
24            case DISCOUNT:
25                service = this.dynamicCostService.select(new DiscountQualifier()).get();
26                break;
27        }
28        return service;
29    }
}
```

DiscountQualifier *selector* elkészítése külön osztályban

```
1 package hu.qwaevisz.inventory.ejbservice.qualified;
2
3 import javax.enterprise.util.AnnotationLiteral;
4
5 public class DiscountQualifier extends
6     AnnotationLiteral<Discount> implements Discount {
7
8     private static final long serialVersionUID = 1L;
9 }
```

DiscountQualifier.java

Érdekesség

Érdekes (lehet) hogy kihasználjuk hogy az annotációkat @interface-ként hozzuk létre (implements Discount).



Részfeladat: Nemzetközi terjeszkedés kapujában az *inventory* elemeket egy külső rendszer is lekérdezi, mely azonban más formátumot vár el, mint amit jelenleg tárolunk és szolgáltatunk. E célra hozzuk létre az `InternationalInventory`-t, melyben megjelenik a termék (`product`: az elem nevének és referenciájának a konkatenációja), illetve az ár (`price`). Utóbbi az értéken kívül a pénznemet is tartalmazza (az árak az *inventory*-ban EUR-ban vannak tárolva).

```
1 {  
2   "price": "23790 HUF",  
3   "product": "LOR78-Lorem20"  
4 }
```

- ▷ Legyen lehetőség az új *bean* kulcsainak programozott konfigurációjára, melyhez ne kelljen a forráskód mezőneveit változtatni (ahogy korábban XML-nél, itt is annotációval vezéreljük a JSON kimenetet).
- ▷ Az aktuális pénznem (példában HUF) egy indítási konfigurációs paraméter legyen, melyet JNDI-ban tároljunk el `java:global/currency` kulccsal, ahogyan az EUR2HUF váltószám is hasonló módon konfigurálható legyen (kulcs: `java:global/exchangeRate`).



InternationalRestService

- ▷ <http://localhost:8080/inventory/api/international/{reference}>
 - A rendszer indítása előtt beállított pénznemben és formátumban visszaadja a referencia számmal adott termék "nemzetközi adatlapját".
- ▷ <http://localhost:8080/inventory/api/international/notify/{reference}>
 - Az előzővel azonos funkcionalitású *endpoint*, de az implementáció során itt eseményeket is fogunk létrehozni. A részfeladatok sorrendhelyes bemutatása végett lesz szétszedve.

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <server xmlns="urn:jboss:domain:1.7">
3   <profile>
4     [..]
5     <subsystem xmlns="urn:jboss:domain:naming:1.4">
6       <remote-naming/>
7       <bindings>
8         <simple name="java:global/currency" value="HUF"/>
9         <simple name="java:global/exchangeRate" value="305"
10           type="int" />
11       </bindings>
12     </subsystem>
13     [..]
14 </profile>
15 </server>
```

Az egyszerű értékek mellett (simple) van lehetőség objektum referenciát is (object-factory) és JNDI *alias*-t is definiálni (lookup).

A változók nevei nem kezdődhetnek akárhogyan. A *container* már meglévő hierarchiájába építi be ezeket, saját elnevezési szabályai szerint. A `java:global` olyan változókat definiál, melyek *scope*-ja a teljes alkalmazás szerverre érvényes.

standalone.xml

Az `InternationalInventoryItemBean` konverziója két szintéren zajlik (ahogy eddig is):

- ▷ Az `InventoryItem`-ből az EJB szolgáltatás réteg `InternationalInventoryItemConverter` CDI *bean*-je készít `InternationalInventoryItemBean`-t.
- ▷ Az `InternationalInventoryItemBean`-ből **Jackson JSON library** segítségével a külső *interface* számára elfogadható JSON példány készül (JAX-RS közbenjárásával).

InternationalInventoryItemBean

```
1 package hu.qwaevisz.inventory.ejbservice.domain;
2
3 import org.codehaus.jackson.annotate.JsonProperty;
4
5 public class InternationalInventoryItemBean {
6
7     private String label;
8     private String price;
9
10    public InternationalInventoryItemBean(String label, String price) { [...] }
11
12    @JsonProperty("product")
13    public String getLabel() { return this.label; }
14
15    public void setLabel(String label) { this.label = label; }
16
17    @JsonProperty("price")
18    public String getPrice() { return this.price; }
19
20    public void setPrice(String price) { this.price = price; }
21
22    @Override
23    public String toString() { return this.label; }
24 }
```

A példa kedvéért szándékosan más nevet használunk a `label` helyett.

A `@JsonProperty` annotációt a *getter* metódusokra helyezhetjük el, és ekvivalens a működése az `@XmlElement(name = "[...]")` annotációval.

InternationalInventoryItemBean.java

JSON Binding függőség

inv-ejbservice / root project (dependencyManagement)



A JAX-B az XML formátumra nézve mindezt már a JavaEE 6 API szintjén támogatja, JSON esetében azonban csupán a Java EE 8 rendelkezik **JSON Binding API**-val (*Jackson*, *Gson*, *Genson* standard API-ját készítették el, a kapcsolódó annotáció: `@JsonProperty`). Ezért szükséges külön függőséget felvenni e célra.

```
1 <project [..]> [..]
2   <properties>
3     <version.jackson>1.9.9</version.jackson>
4   </properties> [..]
5   <dependencyManagement>
6     <dependencies>
7       [..]
8     <dependency>
9       <groupId>org.codehaus.jackson</groupId>
10      <artifactId>jackson-core-asl</artifactId>
11      <version>${version.jackson}</version>
12      <scope>provided</scope>
13    </dependency>
14    <dependency>
15      <groupId>org.codehaus.jackson</groupId>
16      <artifactId>jackson-mapper-asl</artifactId>
17      <version>${version.jackson}</version>
18      <scope>provided</scope>
19    </dependency>
20  </dependencies>
21 </dependencyManagement>
22 </project>
```


InternationalInventoryItemConverter

```
1 package hu.qwaevisz.inventory.ejb.service.converter;
2
3 import
4     hu.qwaevisz.inventory.ejb.service.domain.InternationalInventoryItem;
5 import hu.qwaevisz.inventory.persistence.domain.InventoryItem;
6
7 public interface InternationalInventoryItemConverter {
8     InternationalInventoryItemBean to(InventoryItem item);
9
10 }
```

InternationalInventoryItemConverter.java

Alapértelmezett implementáció

Ahhoz hogy CDI segítségével ennek implementációját *inject*-áljuk, nincs szükségünk semmilyen annotációra. Ha egy implementációja létezik csak, az megkapja automatikusan a `@Default` *qualifier*-t, és az `@Inject` *qualifier* nélküli használata a `@Default` *qualifier*-rel rendelkező implementációt fogja használni.

InternationalInventoryItemConverterImpl

A `@Resource` annotáció a megadott JNDI név alapján kikéri az elemet a JNDI fából. Az `InitialContext`-et ez esetben a *container* biztosítja.

```
1  [...]
2  public class InternationalInventoryItemConverterImpl implements
      InternationalInventoryItemConverter {
3
4      @Resource(lookup = "java:global/currency")
      private String currency;
5
6
7      @Resource(lookup = "java:global/exchangeRate")
      private int exchangeRate;
8
9
10     @Override
11     public InternationalInventoryItemBean to(InventoryItem item) {
12         String label = item.getReference() + "-" + item.getName();
13         String price = item.getValue() * this.exchangeRate + " " +
            this.currency;
14         return new InternationalInventoryItemBean(label, price);
15     }
16 }
```

Qualifier nélküli *injection*

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4     @EJB
5     private InventoryFinder finder;
6
7     @Inject
8     private InternationalInventoryItemConverter internationalConverter;
9
10    @Logged
11    @Override
12    public InternationalInventoryItemBean getInternationalInventoryItem(final String
        reference) throws AdaptorException {
13        return this.internationalConverter.to(this.finder.get(reference));
14    }
15
16 }
```

InventoryFacadeImpl.java



Részfeladat: Adóhatósági okokból jeleznie szükséges a rendszernek, ha egy nemzetközi *inventory* elemet lekérdezik a rendszerből. Ezen értesítésre két - az üzleti szolgáltatástól független - folyamat fog feliratkozni:

- ▷ Egy **riport modul**, mely ez alapján riportot készít (szimuláljuk naplózással)
- ▷ Egy **kezelő modul**, mely jelzi hogy a rendszer mely technikai felhasználója felügyelte a lekérdezést. A technikai felhasználó a szállított alkalmazás konfigurációjával együtt jár (közvetve *inject*-álni fogjuk).

JavaEE **eseménykezeléssel** oldjuk meg a feladatot (ez is a CDI témaköre)!

Első körben egy-egy technikai felhasználót definiáljunk a SANDBOX, TEST és LIVE rendszerekre nézve. Ha a technikai felhasználót biztosító *ClientHolder* CDI *bean*-t *qualifier*-ek segítségével szeretnénk megkülönböztetni, akkor ehhez három *qualifier* annotációra volna szükségünk. Oldjuk meg ezt a problémát **egy *qualifier* és egy *enum* segítségével** (ennek a megoldásnak értelemszerűen ≥ 3 implementáció esetén van jelentősége)!

Technikai kliens

```
1 package hu.qwaevisz.inventory.ejbservice.domain;
2 public class Client {
3
4     private final String reference;
5     private final String name;
6
7     public Client(final String reference, String name) {
8         this.reference = reference;
9         this.name = name;
10    }
11
12    public String getReference() { return this.reference; }
13
14    public String getName() { return this.name; }
15
16    @Override
17    public String toString() {
18        return this.name + " (" + this.reference + ")";
19    }
20 }
```

Client.java

Egy *qualifier* sok helyett

```
1 package hu.qwaevisz.inventory.ejbservice.qualifier;
2 [...]
3 @Qualifier
4 @Retention(RetentionPolicy.RUNTIME)
5 @Target({ ElementType.TYPE, ElementType
6           })
7 public @interface ClientFlag {
8     ClientType value();
9 }
10 }
```

A *qualifier* `value()` "attribútuma" tér vissza a megkülönböztetést hordozó `ClientType` *enum* egy példányával. Így elérjük hogy egy *qualifier* és egy *enum* segítségével N különböző implementációt tudunk kezelni ($N > 2$ esetén érdemes használni).

ClientFlag.java

```
1 package hu.qwaevisz.inventory.ejbservice.domain;
2 public enum ClientType {
3     SANDBOX,
4     LIVE,
5     TEST,
6     CUSTOM
7 }
```

ClientType.java

Technikai klienst biztosító CDI *bean*

```
1 package hu.qwaevisz.inventory.ejb.service.client;
2 import hu.qwaevisz.inventory.ejb.service.domain.Client;
3 public interface ClientHolder {
4     Client getClient();
5 }
```

ClientHolder.java

```
1 public abstract class AbstractClientHolder implements
    ClientHolder {
2
3     private final Client client;
4
5     public AbstractClientHolder(final String reference, String
        name) {
6         this.client = new Client(reference, name);
7     }
8
9     @Override
10    public Client getClient() { return this.client; }
11 }
```

Mindegyik implementáció 1-1 *Client*-et fog létrehozni, annak megfelelő adatokkal amilyen az üzembe állított rendszer. A CDI *bean*, melyet *inject*-álni fogunk biztosítja a technikai felhasználó visszaadását.

AbstractClientHolder.java

Technikai klienseket biztosító implementációk

```
1 @ClientFlag(ClientType.LIVE)
2 public class LiveClientHolder extends AbstractClientHolder {
3     public LiveClientHolder() {
4         super("MCF012", "Matthew C. Flores");
5     }
6 }
```

LiveClientHolder.java

Mindhárom implementáció *inject*-álása egyértelmű, hiszen rendelkeznek *default ctor*-ral. A `@ClientFlag` *qualifier* elhelyezése az osztályon kézenfekvő és egyértelmű megoldás.

```
1 @ClientFlag(ClientType.SANDBOX)
2 public class SandboxClientHolder extends AbstractClientHolder {
3     public SandboxClientHolder() {
4         super("SVW987", "Scott V. Wright");
5     }
6 }
```

SandboxClientHolder.java

```
1 @ClientFlag(ClientType.TEST)
2 public class TestClientHolder extends AbstractClientHolder {
3     public TestClientHolder() {
4         super("TEST", "Test System");
5     }
6 }
```

TestClientHolder.java

- ▷ Esemény
 - POJO
 - NotifierEvent
- ▷ Eseménykezelő(k)
 - ReportEventHandler és NotifierEventHandler
 - @Observes NotifierEvent (CDI annotáció)
 - Akármennyi eseménykezelő feliratkozhat egy eseményre
- ▷ Feliratkozás, regisztrálás, eseményküldés

- ```
1 @Inject
2 private Event<NotifierEvent> notifier;
3 [...]
4 this.notifier.fire(new NotifierEvent([..]));
```

# NotifierEvent

## Esemény

```
1 package hu.qwaevisz.inventory.ejb.service.event;
2 [...]
3 public class NotifierEvent implements Serializable {
4
5 private InternationalInventoryItemBean bean;
6
7 public NotifierEvent() {}
8
9 public NotifierEvent(InternationalInventoryItemBean bean) {
10 this.bean = bean;
11 }
12
13 public InternationalInventoryItemBean getBean() {
14 return this.bean;
15 }
16
17 [...]
18 }
```

NotifierEvent.java

# ReportEventHandler

Eseménykezelő

```
1 package hu.qwaevisz.inventory.ejbsservice.event;
2 [...]
3 public class ReportEventHandler implements Serializable {
4
5 private static final Logger LOGGER =
6 Logger.getLogger(ReportEventHandler.class);
7
8 public void handle(@Observes NotifierEvent event) {
9 LOGGER.info("Report this event: " + event);
10 }
11 }
```

ReportEventHandler.java

# NotifierEventHandler

Eseménykezelő

```
1 package hu.qwaevisz.inventory.ejb.service.event;
2 [...]
3 public class NotifierEventHandler implements Serializable {
4
5 private static final Logger LOGGER =
6 Logger.getLogger(NotifierEventHandler.class);
7
8 @Inject
9 @ClientFlag(ClientType.LIVE)
10 private ClientHolder clientHolder;
11
12
13
14 public void handle(@Observes NotifierEvent event) {
15 LOGGER.info("Get " + event.getBean() + " from the Inventory
16 system. Technical user: " +
17 this.clientHolder.getClient());
18 }
19 }
```

Ez az `ClientHolder` *inject* határozza meg hogy a *deployment* melyik technikai felhasználó nevében üzemel (a szimulált feladat értelmezésében, valójában itt nem végzünk hasznos műveletet).

# Esemény kiváltása

inv-ejbservice project

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4 @EJB
5 private InventoryFinder finder;
6
7 @Inject
8 private InternationalInventoryItemConverter internationalConverter;
9
10 @Inject
11 private Event<NotifierEvent> notifier;
12
13 @Override
14 public InternationalInventoryItemBean getInternationalInventoryItemWithEvent(String
15 reference) throws AdaptorException {
16 final InternationalInventoryItemBean bean =
17 this.internationalConverter.to(this.finder.get(reference));
18 this.notifier.fire(new NotifierEvent(bean));
19 return bean;
20 }
21 }
```

InventoryFacadeImpl.java

# Produces

Factory beiktatása az *injection* folyamatába

## Probléma

A most bemutatott `ClientHolder` példa természetesen "szimulációja" csupán annak az esetnek, hogy miként lehet sok CDI *bean* implementációt megkülönböztetni elegáns módon, feleslegesen sok *qualifier* létrehozása nélkül. Ugyanezen "szimuláció" során felvethetjük a következő problémát: elég körülményes a kódban az `@Inject`-et módosítani (ehhez újra kell fordítani a kódot). Miként tudnánk **paraméteres `ClientHolder` implementációt** készíteni, ahol a paraméter pl. a már megismert JNDI *tree*-ben lenne inicializálható.

Ezen új implementációt lássuk el egy `CUSTOM ClientFlag`-gel, de természetesen itt megjegyzendők az alábbiak:

- ▷ A `CUSTOM` paraméterezhető `ClientHolder` feleslegessé teszi a többit.
- ▷ A paraméteres `ctor`-ral rendelkező CDI *bean*-ek *inject*-álása és a *qualifier* + *enum* párosítás nincs kapcsolatban, mindegyik módszer a másiktól függetlenül is alkalmazható.

A paraméteres CDI *bean*-ekkel az a baj, hogy *container* nem tudja kitalálni a paraméter értékét. A megoldás egy **factory létrehozása**, melyben ezen létrehozás szabályát határozzuk meg. Paraméteres *inject*-hez a `javax.enterprise.inject.Produces` annotációt fogjuk használni<sup>2</sup>.

<sup>2</sup> Ne keverjük össze a `javax.enterprise.inject.Produces` és a `javax.ws.rs.Produces` annotációkat!

# CustomClientHolder

```
1 package hu.qwaevisz.inventory.ejbservice.client;
2
3 public class CustomClientHolder extends AbstractClientHolder {
4
5 public CustomClientHolder(final String name) {
6 super(name.substring(0, 3).toUpperCase(), name);
7 }
8
9 }
```

CustomClientHolder.java

Nem tehetjük az osztályra a `@ClientFlag(ClientType.CUSTOM)` annotációt, mert nem tudja a *container* a *name constructor* paraméter értékét! *Factory*-ra van szükség!

# ClientHolderFactory

```
standalone.xml:
<simple name="java:global/customClientName"
value="Dolores M. Putto" />
```

A negyedik típusú `ClientHolder`-t *factory* segítségével hozzuk létre. Ehhez egy "singleton" osztályt készítünk (`@ApplicationScoped` annotáció), melyben egy `@Produces` + *qualifier* annotációval ellátott metódus adja vissza az ebben létrehozott `ClientHolder` példányt. A `@Produces` annotációval ellátott elemeket a *container* összegyűjti, és egy adott típusú erőforrás kérésekor használja őket (legtöbbször csak egyszer futnak le).

Felhasználás:

```
1 package hu.qwaevisz.inventory.ejbser
2 [...]
3 @ApplicationScoped
4 public class ClientHolderFactory {
5
6 @Resource(lookup = "java:global/customClientName")
7 private String clientName;
8
9 @Produces
10 @ClientFlag(ClientType.CUSTOM)
11 public ClientHolder getCustomClientHolder() {
12 return new CustomClientHolder(this.clientName);
13 }
14 }
```

```
1 @Inject
2 @ClientFlag(ClientType.CUSTOM)
3 private ClientHolder clientHolder;
```





**Részfeladat:** Az *inventory* rendszerünk egy rendkívül hasznos új funkciója, hogy **véletlen számot** kérésre **vissza tud adni**. Mindez azért remek, mert így okot ad arra, hogy véletlen számot injectáljunk CDI segítségével. A feladat valójában arról szól, hogy miként injectálunk úgy egy erőforrást, hogy az minden ismételt elővétel során újból létrejöjjön (a létrehozási szakasz azért használ véletlen számot ez esetben, hogy látványos legyen az új futás eredménye).

A feladat során azt is megvizsgáljuk, hogyan *injectálunk* egyszerű értékeket (számokat), amiben nem a szintaktika, hanem sokkal inkább a szemlélet lesz a lényeges.



## ConfigurationRestService

- ▷ <http://localhost:8080/inventory/api/configuration/numbers/{quantity}>
  - A megadott mennyiségnek megfelelő véletlen számot generál 0 és MAX között, ahol a MAX értékét JNDI *tree*-n keresztül konfiguráljuk (JNDI név: `java:global/maxLength`).
- ▷ <http://localhost:8080/inventory/api/configuration/host>
  - Visszaadja az alkalmazás `host` konfigurációs értékét (maga a név teljesen lényegtelen). Eredetileg a `host` értéke JNDI fában található meg (JNDI név: `java:global/inventoryhost`).

# Maximum érték mint erőforrás

```
1 package hu.qwaevisz.inventory.ejbservice.qualifier;
2 [..]
3 @Qualifier
4 @Retention(RetentionPolicy.RUNTIME)
5 @Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,
6 ElementType.PARAMETER })
7 public @interface MaxNumber {
8 }
```

## MaxNumber.java

```
1 package hu.qwaevisz.inventory.ejbservice.service;
2 [..]
3 @ApplicationScoped
4 public class MaxNumberFactory {
5
6 @Resource(lookup = "java:global/maxNumber")
7 private int maxValue;
8
9 @Logged
10 @Produces
11 @MaxNumber
12 public int getMaxNumber() {
13 return this.maxValue;
14 }
15 }
```

**Fontos!** Ahhoz hogy ez az érték frissüljön az alkalmazásban, az alkalmazás szervert újra kell indítani! Változtatható konfigurációs paraméterre pl. a JMX jobb megoldás lehet.

## MaxNumberFactory.java

# Kell ez a komplexitás?

## Normálisak vagyunk?

Egy `MAX` konstans létrehozása helyett két új típust vezetünk be a rendszerbe? Ezt nehéz szépíteni, szintaktikailag valóban erről van szó. Azonban ez a `@MaxNumber` erőforrásként jelenik meg, így ha később pl. nem konstans lesz, akkor úgy tudjuk az alkalmazás ezen részét átalakítani, hogy semmilyen más komponenst nem érintünk.

# JNDI változó állítása JBoss CLI-on keresztül



```
java:global/maxLength
```

```
1 > jboss-cli --connect
2 [/] /subsystem=naming/binding=java\:global\/maxLength:read-resource
3 {
4 "outcome" => "success",
5 "result" => {
6 "binding-type" => "simple",
7 "cache" => undefined,
8 "class" => undefined,
9 "environment" => undefined,
10 "lookup" => undefined,
11 "module" => undefined,
12 "type" => "int",
13 "value" => "100"
14 }
15 }
16 [/]
17 /subsystem=naming/binding=java\:global\/maxLength:write-attribute(name=value,value=200
18 {
19 "outcome" => "success",
20 "response-headers" => {
21 "operation-requires-reload" => true,
22 "process-state" => "reload-required"
23 }
24 }
```

A JNDI névből a : és a / karaktert escape-elni kell. Az olvasó *operation* neve **read-resource**, míg az író művelet neve **write-attribute**.

**Fontos!** A fenti módszerrel ellenőrizhető, hogy a JNDI paramétereket az alkalmazáson belül nem tudjuk "átírni", reload-olni. Akkor sem, ha InitialContext-ből kérjük ki. A szerver újraindítása szükséges.

A JNDI fát a JBoss *webconsole*-on

(<http://localhost:9990/console/App.html#naming>) is tudjuk böngészni (Runtime | JNDI view).

# Véletlen szám mint erőforrás

```
1 [..]
2 @Qualifier
3 @Retention(RetentionPolicy.RUNTIME)
4 @Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,
5 ElementType.PARAMETER })
6 public @interface Random {}
```

## Random.java

```
1 package hu.qwaevisz.inventory.ejbservice.service;
2 [..]
3 @ApplicationScoped
4 public class RandomNumberFactory {
5
6 private final java.util.Random random = new
7 java.util.Random(System.currentTimeMillis());
8
9 @Inject
10 @MaxNumber
11 private int maxNumber;
12
13 @Logged
14 @Produces
15 @Random
16 public int next() {
17 return this.random.nextInt(this.maxNumber);
18 }
19 }
```

A `@MaxNumber` egyszer lesz *injectálva*, mivel egyszerűen egy `int` érték szerepel az `@Inject` mellett. Ugyanazzal a módszerrel a `@Random` szám factory-ja (`next()` metódus) is csupán egyszer futna le, így itt más módszerre lesz szükség *injection* során.

# Véletlen szám erőltetett *inject*-álása

inv-ejbsservice project

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4 @Inject
5 @Random
6 private Instance<Integer> randomNumber;
7
8 @Logged
9 @Override
10 public List<Integer> getRandomNumbers(final int quantity) throws AdaptorException {
11 final List<Integer> result = new ArrayList<>();
12 for (int i = 0; i < quantity; i++) {
13 result.add(this.randomNumber.get());
14 }
15 return result;
16 }
17
18 }
```

InventoryFacadeImpl.java

A generikus `Instance<T>` típus esetén a `T` helyére írjuk az *inject*-állandó dinamikus erőforrás típusát, és a `T` `get()` metódus meghívásakor az inicializálás mindig ismételten meg fog történni. Ugyanezt a generikus típus használtuk a CDI *bean*-ek dinamikus kiválasztásakor is, de ott a `get()` helyett a `select()` metódust alkalmaztuk és ismertük meg.



Az alkalmazásnak számos konfigurációs paramétere lehet, melyeket pl. JNDI fában tárolunk. Ilyen értékek pl. a

- ▷ `java:global/inventoryhost`, és a
- ▷ `java:global/currency`

**Részfeladat:** Mindegyik számára létrehozhatnánk egy *qualifier*-t, melynek segítségével bárhova elhelyezhetjük, azonban ez sok paraméter esetén kényelmetlen lehet. Megoldást a `@Named( String )` annotáció jelenthet, mely használata során nem kell mindig új és új annotációkat létrehozni ilyen esetekben.



# InventoryJndiPropertyProvider

```
1 package hu.qvaevisz.inventory.ejbservice.service;
2 [...]
3 public class InventoryJndiPropertyProvider {
4
5 @Resource(lookup = "java:global/inventoryhost")
6 private String inventoryHost;
7
8 @Resource(lookup = "java:global/currency")
9 private String currency;
10
11 @Produces
12 @Named("host")
13 public String getInventoryHost() {
14 return this.inventoryHost;
15 }
16
17 @Produces
18 @Named("currency")
19 public String getCurrency() {
20 return this.currency;
21 }
22 }
```

Saját *qualifier* helyett a `@Named` paraméteres annotációt használjuk. Csupán ennyi a különbség az eddigiekhez képest.

InventoryJndiPropertyProvider.java

# InventoryConfiguration

```
1 package hu.qwaevisz.inventory.ejbservice.service;
2
3 public class InventoryConfiguration {
4
5 private final String host;
6 private final String currency;
7
8 public InventoryConfiguration(String host, String currency) {
9 this.host = host;
10 this.currency = currency;
11 }
12
13 public String getHost() {
14 return this.host;
15 }
16
17 public String getCurrency() {
18 return this.currency;
19 }
20 }
```

Hozzunk létre egy osztályt, mely a konfigurációs paramétereket tárolja. Az osztálynak nincs alapértelmezett *constructor*-a, így csak *factory* és *@Produces* segítségével tudjuk majd *inject*-álni.

InventoryConfiguration.java

# InventoryConfigurationProvider

```
1 package hu.qwaevisz.inventory.ejbservice.service;
2
3 import javax.enterprise.inject.Produces;
4 import javax.inject.Named;
5
6 public class InventoryConfigurationProvider {
7
8 @Produces
9 public InventoryConfiguration config(@Named("host") String host, @Named("currency")
10 String currency) {
11 return new InventoryConfiguration(host, currency);
12 }
13 }
```

## InventoryConfigurationProvider.java

A *qualifier*-eket nem csupán az `@Inject` mellett használhatjuk (bár ugyanezen *factory*-t így is el lehetne készíteni). Paraméteres `@Produces` metódusok esetén kényelmes és tiszta megoldás ha az argumentumokat annotáljuk meg.

# Konfigurációs paraméter használata

inv-ejb-service project

```
1 @Stateless(mappedName = "ejb/inventoryFacade")
2 public class InventoryFacadeImpl implements InventoryFacade {
3
4 @Inject
5 private InventoryConfiguration configuration;
6
7 @Override
8 public String getHost() throws AdaptorException {
9 return this.configuration.getHost();
10 }
11
12 }
```

InventoryFacadeImpl.java

Típus alapján a CDI bean megtalálható lesz, lefut a `@Produces` *factory*-ja, melybe *inject*-álódnak a `@Named` paraméterek, melyeknek *factory*-ja egy-egy JNDI értékből inicializálódik.

Mindennek természetesen egy nagyvállalati rendszernél van elsősorban szerepe, nem mindenhol indokolt a felelősséget ennyire szétosztani.