Inventory #gradle

Interceptor, JNDI variable, CDI, JSON Binding

Óbudai Egyetem, Java Enterprise Edition

Műszaki Informatika szak

Labor 9

Bedők Dávid

2018-03-14

v1.0

Inventory

JavaEE további lehetőségei

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`

Project felépítése

Inventory alkalmazás

- **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`

Adat réteg

Inventory alkalmazás

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.

Copyright információk

Részfeladat

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

Eredeti JSON

```
{
    "reference": "LOR78",
    "name": "Lorem20",
    "type": "BOOK",
    "value": 78
}
```

Módosított JSON

```
{
    "reference": "LOR78",
    "name": "Lorem20 @ 2018 Inventory",
    "type": "BOOK",
    "value": 78
}
```

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 felelősségek 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

Mindezek 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* végett[1]).
- **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
`javax.interceptor.AroundInvoke`	A metódus megjelölése interceptor-ként.
`javax.interceptor.AroundTimeout`	Enterpise bean timeout esetén a metódust time-out interceptor-ként jelöli meg.
`javax.annotation.PostConstruct`	A bean létrehozás életciklus interceptora.
`javax.annotation.PreDestroy`	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.CopyrightIntercepto
7   </interceptor-class>
8   <method-name>getInventory</method-name>
9 </interceptor-binding>
```

ejb-jar.xml

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

Copyright információk tesztelése

RESTful webszolgáltatáson keresztül

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 }
```

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

- Az `@Interceptor` (egyes szám) annotáció segítségével megjelöljük az *interceptor* osztályt.
- Létrehozzunk 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

```java
public interface InventoryItemBeanConverter {
  InventoryItemBean to(InventoryItem item);
  List<InventoryItemBean> to(List<InventoryItem> items);
}
```

Nincs az interface-en @Local annotáció!

InventoryItemBeanConverter.java

```java
public class InventoryItemBeanConverterImpl implements InventoryItemBeanConverter {

  @Override
  public InventoryItemBean to(InventoryItem item) { [..] }

  @Override
  public List<InventoryItemBean> to(List<InventoryItem> items) {
    return items.stream().map(this::to).collect(Collectors.toList());
  }
}
```

Nincs az osztályon @Stateless annotáció!

InventoryItemBeanConverterImpl.java

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements Inve

  @Inject
  private InventoryItemBeanConverter converter;

  [..]
}
```

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
Container szolgáltatásokat biztosít (tranzakció kezeléssel, *security*-val, konkurrencia kezeléssel, stb.)	Mudularizációt, *separation of concerns*-t, eseménykezelést, *injection*-t biztosít.
Az EJB-k CDI bean-ek is, ennek minden előnyével.	-
`@Stateful`, `@Stateless`, `@Singleton` felbontás	`@RequestScoped`, `@SessionScoped`, `@ApplicationScoped`, `@ConversationScoped` (JSF) vagy custom *scope* 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 *container managed* 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*).

Naplózás *interceptor* segítségével

Részfeladat

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
           http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
5
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

```java
@Interceptor
public class LoggedInterceptor implements Serializable {

  private static final Logger LOGGER = Logger.getLogger(LoggedInterceptor.class);

  @AroundInvoke
  public Object logMethodInvocations(InvocationContext context) throws Exception {
    final StringBuilder info = new StringBuilder();
    info.append(context.getTarget().getClass().getName()).append(".").append(context.getMet
    final Object[] parameters = context.getParameters();
    if (parameters != null) {
      int i = parameters.length - 1;
      for (final Object parameter : parameters) {
        info.append(parameter.toString());
        if (i > 0) {
          info.append(",");
        }
        i--;
      }
    }
    info.append(")");
    LOGGER.info("Entering: " + info.toString());
    final long start = System.currentTimeMillis();
    final Object result = context.proceed();
    final long end = System.currentTimeMillis();
    LOGGER.info("Exiting: " + info.toString() + " - running time: " + (end - start) + "
        millisecond(s)");
    return result;
  }
}
```

LoggedInterceptor.java

Logged

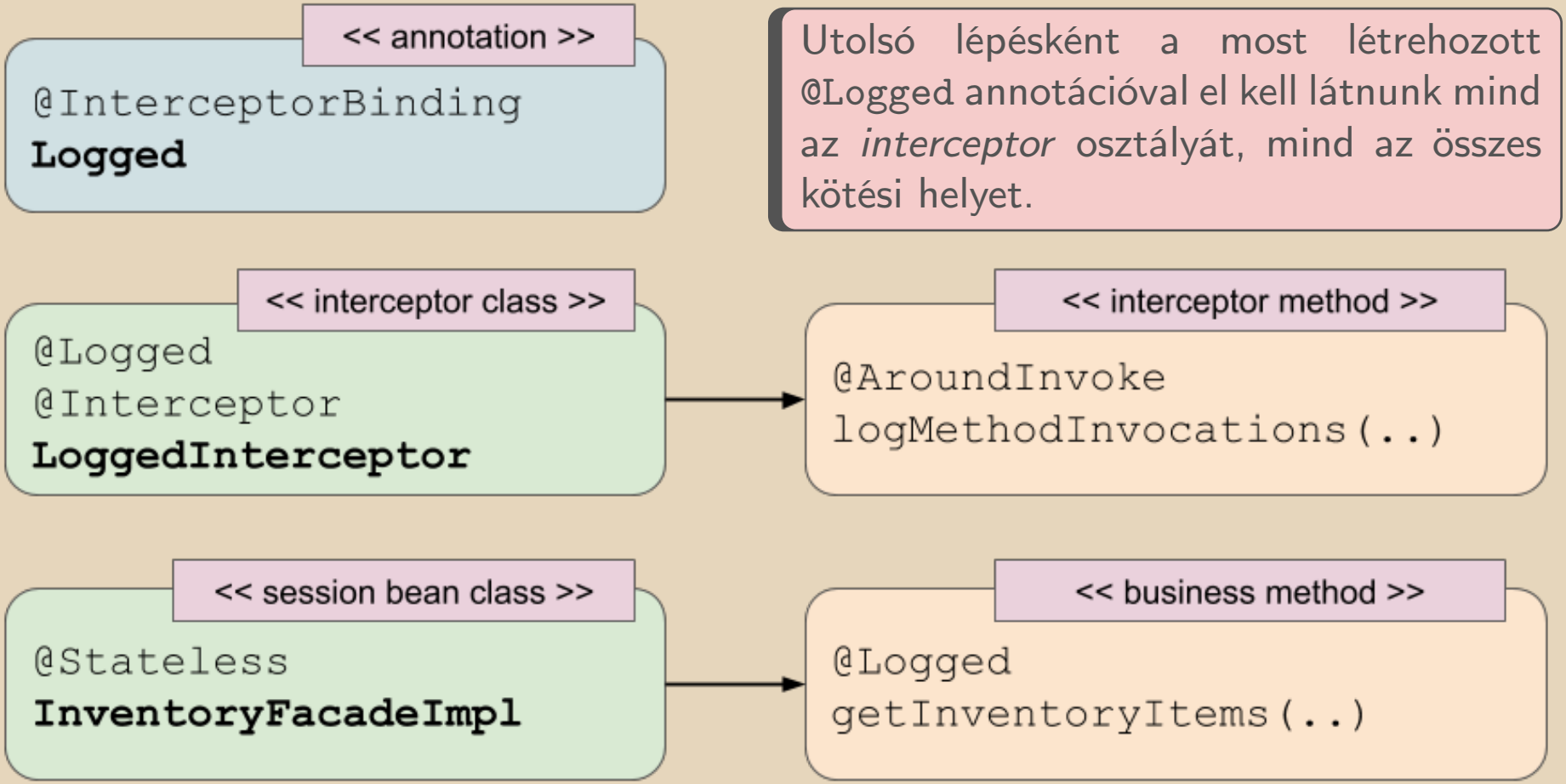
@InterceptorBinding annotáció

```
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

Áttekintés


<< 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)
         throws Exception {
10     [..]
11   }
12
13 }
```

LoggedInterceptor.java

Interceptor bekötése CDI segítségével

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {

  @EJB
  private InventoryFinder finder;

  @Inject
  private InventoryItemBeanConverter converter;

  @Logged
  @Override
  @Interceptors({ CopyrightInterceptor.class })
  public List<InventoryItemBean> getInventoryItems(final InventoryType type) throws
       AdaptorException {
    return this.converter.to(this.finder.list(type));
  }

  @Inject
  private InventorySearch search;

  @Logged
  @Override
  public List<InventoryItemBean> getInventoryItems(final InventoryType type, final
       String nameTerm) throws AdaptorException {
    return this.converter.to(this.search.list(type, nameTerm));
  }

  [..]
}
```

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

InventoryFacadeImpl.java

Naplózás és futási idő tesztelése - I

RESTful webszolgáltatáson keresztül

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 ]
```

Promóciós időszak aktiválása *qualifier* segítségével

Részfeladat

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 kellenek *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

```java
package hu.qwaevisz.inventory.ejbservice.qualifier;
[..]
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD,
    ElementType.FIELD, ElementType.PARAMETER })
public @interface Standard {

}
```

Standard.java

```java
package hu.qwaevisz.inventory.ejbservice.qualifier;
[..]
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD,
    ElementType.FIELD, ElementType.PARAMETER })
public @interface Discount {

}
```

Discount.java

CostService és implementációi

```java
package hu.qwaevisz.inventory.ejbservice.cost;
public interface CostService {
  int getCost(int originalValue);
}
```

CostService.java

```java
[..]
@Standard
public class NormalCostService implements CostService {
  @Override
  public int getCost(int originalValue) {
    return originalValue;
  }
}
```

NormalCostService.java

```java
[..]
@Discount
public class PromotionalCostService implements CostService {
  private static final float DISCOUNT_PERCENT = 20;
  @Override
  public int getCost(int originalValue) {
    return Math.round(originalValue * (1 - (DISCOUNT_PERCENT / 100)));
  }
}
```

PromotionalCostService.java

Qualifier alapú *injection*

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {

  @EJB
  private InventoryFinder finder;

  @Inject
  private InventoryItemBeanConverter converter;

  @Inject
  @Discount
  private CostService costService;

  @Logged
  @Override
  public InventoryItemBean buyInventoryItem(String reference) throws AdaptorException {
    final InventoryItemBean bean = this.converter.to(this.finder.get(reference));
    bean.setValue(this.costService.getCost(bean.getValue()));
    return bean;
  }
}
```

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*

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {
  [..]
  @Any
  @Inject
  private Instance<CostService> dynamicCostService;

  @Logged
  @Override
  public InventoryItemBean buyInventoryItem(String reference, PricingStrategy pricing)
      throws AdaptorException {
    final InventoryItemBean bean = this.converter.to(this.finder.get(reference));
    bean.setValue(this.getCostService(pricing).getCost(bean.getValue()));
    return bean;
  }
  private CostService getCostService(PricingStrategy pricing) {
    CostService service = null;
    switch (pricing) {
      case STANDARD:
        service = this.dynamicCostService.select(new AnnotationLiteral<Standard>() {
          private static final long serialVersionUID = 1L;
        }).get();
        break;
      case DISCOUNT:
        service = this.dynamicCostService.select(new DiscountQualifier()).get();
        break;
    }
    return service;
  }
}
```

InventoryFacadeImpl.java

DiscountQualifier *selector* elkészítése külön osztályban

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

JNDI változó definiálása

JBoss

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

```
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"
               type="int" />
10       </bindings>
11     </subsystem>
12     [..]
13   </profile>
14   [..]
15 </server>
```

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`

Konverzió

InternationalInventoryItemBean

Az InternationalInventoryItemBean konverziója két színtéren 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

```java
package hu.qwaevisz.inventory.ejbservice.domain;

import org.codehaus.jackson.annotate.JsonProperty;

public class InternationalInventoryItemBean {

  private String label;
  private String price;

  public InternationalInventoryItemBean(String label, String price) { [..] }

  @JsonProperty("product")
  public String getLabel() { return this.label; }

  public void setLabel(String label) { this.label = labe

  @JsonProperty("price")
  public String getPrice() { return this.price;}

  public void setPrice(String price) { this.price = pric

  @Override
  public String toString() { return this.label; }
}
```

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 project

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: @JsonbProperty). Ezért szükséges külön függőséget felvenni e célra.

```
1 dependencies {
2   [..]
3   compile group: 'org.codehaus.jackson', name: 'jackson-core-asl', version:
        jacksonVersion
4   compile group: 'org.codehaus.jackson', name: 'jackson-mapper-asl', version:
        jacksonVersion
5   [..]
6 }
```

inv-ejbservice | build.gradle

```
1 [..]
2 ext {
3   [..]
4   jacksonVersion = '1.9.9'
5   [..]
6 }
7 [..]
```

root | build.gradle

InternationalInventoryItemConverter

```
1  package hu.qwaevisz.inventory.ejbservice.converter;
2
3  import
       hu.qwaevisz.inventory.ejbservice.domain.InternationalInventoryIte
4  import hu.qwaevisz.inventory.persistence.domain.InventoryItem;
5
6  public interface InternationalInventoryItemConverter {
7
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")
5    private String currency;
6
7    @Resource(lookup = "java:global/exchangeRate")
8    private int exchangeRate;
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 }
```

InternationalInventoryItemConverterImpl.java

Qualifier nélküli *injection*

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {

  @EJB
  private InventoryFinder finder;

  @Inject
  private InternationalInventoryItemConverter internationalConverter;

  @Logged
  @Override
  public InternationalInventoryItemBean getInternationalInventoryItem(final String
       reference) throws AdaptorException {
    return this.internationalConverter.to(this.finder.get(reference));
  }

}
```

InventoryFacadeImpl.java

Részfeladat: Adóhatósági okokból jeleznie szükséges a rendszernek, ha egy nemzetközi *inventory* elemet lekérdeznek 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

```java
package hu.qwaevisz.inventory.ejbservice.domain;
public class Client {

  private final String reference;
  private final String name;

  public Client(final String reference, String name) {
    this.reference = reference;
    this.name = name;
  }

  public String getReference() { return this.reference; }

  public String getName() { return this.name; }

  @Override
  public String toString() {
    return this.name + " (" + this.reference + ")";
  }
}
```

Client.java

Egy *qualifier* sok helyett

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

```java
package hu.qwaevisz.inventory.ejbservice.client;
import hu.qwaevisz.inventory.ejbservice.domain.Client;
public interface ClientHolder {
  Client getClient();
}
```

ClientHolder.java

```java
public abstract class AbstractClientHolder imp
    ClientHolder {

  private final Client client;

  public AbstractClientHolder(final String reference, String
      name) {
    this.client = new Client(reference, name);
  }

  @Override
  public Client getClient() { return this.client; }
}
```

AbstractClientHolder.java

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.

Technikai klienseket biztosító implementációk

```java
@ClientFlag(ClientType.LIVE)
public class LiveClientHolder extends AbstractClientHolder {
  public LiveClientHolder() {
    super("MCF012", "Matthew C. Flores");
  }
}
```

LiveClientHolder.java

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

```java
@ClientFlag(ClientType.SANDBOX)
public class SandboxClientHolder extends AbstractClientHolder {
  public SandboxClientHolder() {
    super("SVW987", "Scott V. Wright");
  }
}
```

SandboxClientHolder.java

```java
@ClientFlag(ClientType.TEST)
public class TestClientHolder extends AbstractClientHolder {
  public TestClientHolder() {
    super("TEST", "Test System");
  }
}
```

TestClientHolder.java

Eseménykezelés

JavaEE CDI környezetben

- Esemény
  - POJO
  - `NotifierEvent`
- Eseménykezelő(k)
  - `ReportEventHandler` és `NotifierEventHandler`
  - `@Observes NotifierEvent` (CDI annotáció)
  - Akármennyi eseménykezelő feliratkozhat egy eseményre
- Feliratkozás, regisztrálás, eseménykküldés
  -
```
1 @Inject
2 private Event<NotifierEvent> notifier;
3 [..]
4 this.notifier.fire(new NotifierEvent([..]));
```

NotifierEvent

Esemény

```
1  package hu.qwaevisz.inventory.ejbservice.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

```java
package hu.qwaevisz.inventory.ejbservice.event;
[..]
public class ReportEventHandler implements Serializable {

  private static final Logger LOGGER =
      Logger.getLogger(ReportEventHandler.class);

  public void handle(@Observes NotifierEvent event) {
    LOGGER.info("Report this event: " + event);
  }

}
```

ReportEventHandler.java

NotifierEventHandler

Eseménykezelő

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

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

NotifierEventHandler.java

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
         reference) throws AdaptorException {
15     final InternationalInventoryItemBean bean =
           this.internationalConverter.to(this.finder.get(reference));
16     this.notifier.fire(new NotifierEvent(bean));
17     return bean;
18   }
19 }
```

InventoryFacadeImpl.java

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 megjegyzendőek 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[2].

[2] Ne keverjük össze a `javax.enterprise.inject.Produces` és a `javax.ws.rs.Produces` annotációkat!

CustomClientHolder

```java
package hu.qwaevisz.inventory.ejbservice.client;

public class CustomClientHolder extends AbstractClientHolder {

  public CustomClientHolder(final String name) {
    super(name.substring(0, 3).toUpperCase(), name);
  }

}
```

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

```
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 }
```

ClientHolderFactory.java

Véletlenszám generátor

Részfeladat

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/maxNumber).
- 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

```java
package hu.qwaevisz.inventory.ejbservice.qualifier;
[..]
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,
     ElementType.PARAMETER })
public @interface MaxNumber {

}
```

MaxNumber.java

```java
package hu.qwaevisz.inventory.ejbservice.service;
[..]
@ApplicationScoped
public class MaxNumberFactory {

  @Resource(lookup = "java:global/maxNumber")
  private int maxValue;

  @Logged
  @Produces
  @MaxNumber
  public int getMaxNumber() {
    return this.maxValue;
  }
}
```

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/maxNumber

```
> jboss-cli --connect
[/] /subsystem=naming/binding=java\:global\/maxNumber:read-resource
{
    "outcome" => "success",
    "result" => {
        "binding-type" => "simple",
        "cache" => undefined,
        "class" => undefined,
        "environment" => undefined,
        "lookup" => undefined,
        "module" => undefined,
        "type" => "int",
        "value" => "100"
    }
}
[/] /subsystem=naming/binding=java\:global\/maxNumber:write-attribute(name=value,value=200
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

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,
       ElementType.PARAMETER })
5  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
         java.util.Random(System.currentTimeMillis());
7
8    @Inject
9    @MaxNumber
10   private int maxNumber;
11
12   @Logged
13   @Produces
14   @Random
15   public int next() {
16     return this.random.nextInt(this.maxNumber);
17   }
18
19 }
```

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

RandomNumberFactory.java

Véletlen szám erőltetett *inject*-álása

inv-ejbservice project

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {

  @Inject
  @Random
  private Instance<Integer> randomNumber;

  @Logged
  @Override
  public List<Integer> getRandomNumbers(final int quantity) throws AdaptorException {
    final List<Integer> result = new ArrayList<>();
    for (int i = 0; i < quantity; i++) {
      result.add(this.randomNumber.get());
    }
    return result;
  }

}
```

InventoryFacadeImpl.java

A generikus `Instance<T>` típus esetén a `T` helyére írjuk az *inject*-álandó dinamikus erőforrás típusát, és a 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.

Konfigurációs paraméterek

Részfeladat

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

```java
package hu.qwaevisz.inventory.ejbservice.service;
[..]
public class InventoryJndiPropertyProvider {

  @Resource(lookup = "java:global/inventoryhost")
  private String inventoryHost;

  @Resource(lookup = "java:global/currency")
  private String currency;

  @Produces
  @Named("host")
  public String getInventoryHost() {
    return this.inventoryHost;
  }

  @Produces
  @Named("currency")
  public String getCurrency() {
    return this.currency;
  }
}
```

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

InventoryJndiPropertyProvider.java

InventoryConfiguration

```java
package hu.qwaevisz.inventory.ejbservice.service;

public class InventoryConfiguration {

  private final String host;
  private final String currency;

  public InventoryConfiguration(String host, String currency) {
    this.host = host;
    this.currency = currency;
  }

  public String getHost() {
    return this.host;
  }

  public String getCurrency() {
    return this.currency;
  }
}
```

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

```java
package hu.qwaevisz.inventory.ejbservice.service;

import javax.enterprise.inject.Produces;
import javax.inject.Named;

public class InventoryConfigurationProvider {

  @Produces
  public InventoryConfiguration config(@Named("host") String host, @Named("currency")
      String currency) {
    return new InventoryConfiguration(host, currency);
  }

}
```

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-ejbservice project

```java
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {

  @Inject
  private InventoryConfiguration configuration;

  @Override
  public String getHost() throws AdaptorException {
    return this.configuration.getHost();
  }

}
```

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.