Java Enterprise Edition 6

The next evolution

CONSULTING | ICT | OUTSOURCING

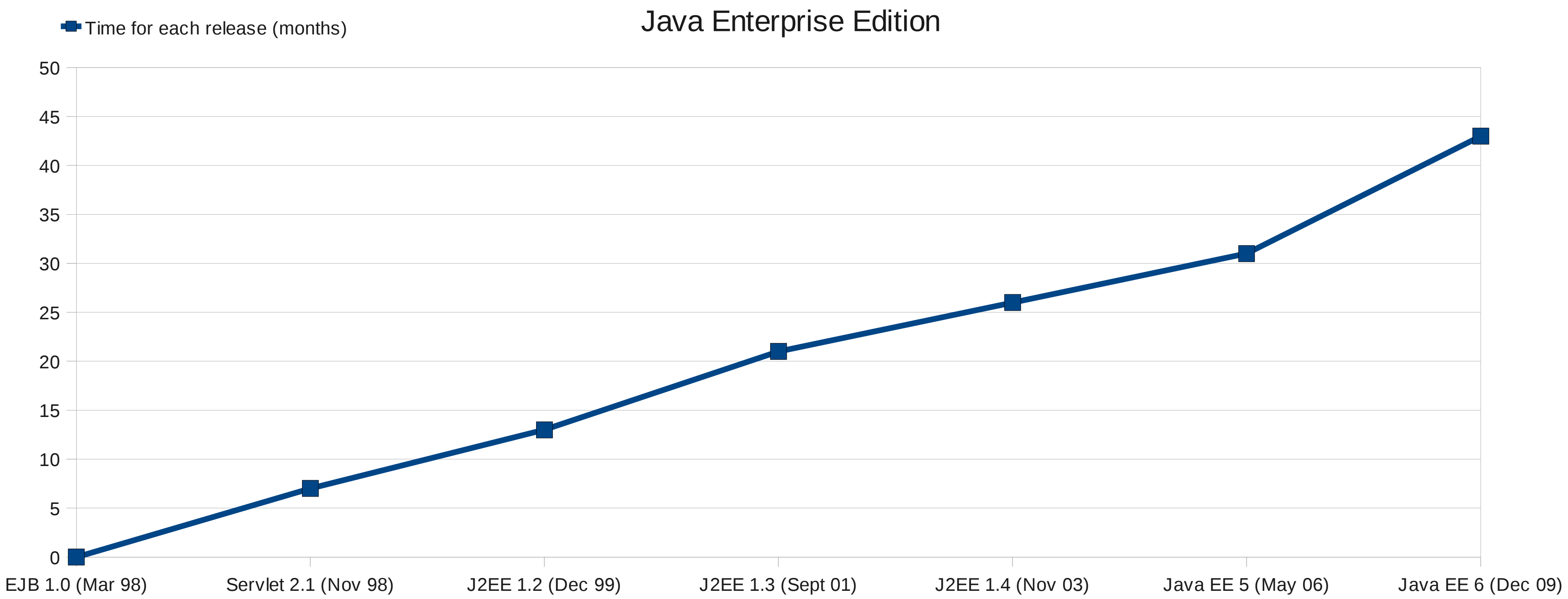
ORDINA.OPGELOST.

Agenda

• Introduction J2EE and Java EE	
• Contexts and Dependency Injection	(JSR-299)
• Web Tier	(JSR-314)
• Enterprise JavaBeans 3.1	(JSR-318)
• Persistence	(JSR-317)
• Bean Validation	(JSR-303)

Java EE - Snelheid

- Every 3 years a new version



Java 2 Enterprise Edition

J2EE

- Heavyweight
- Big enemy of the productive developer
- Impossible to unit test

Java EE 5

- Ease of development
- JavaServer Faces
- EJB 3.0
- JPA

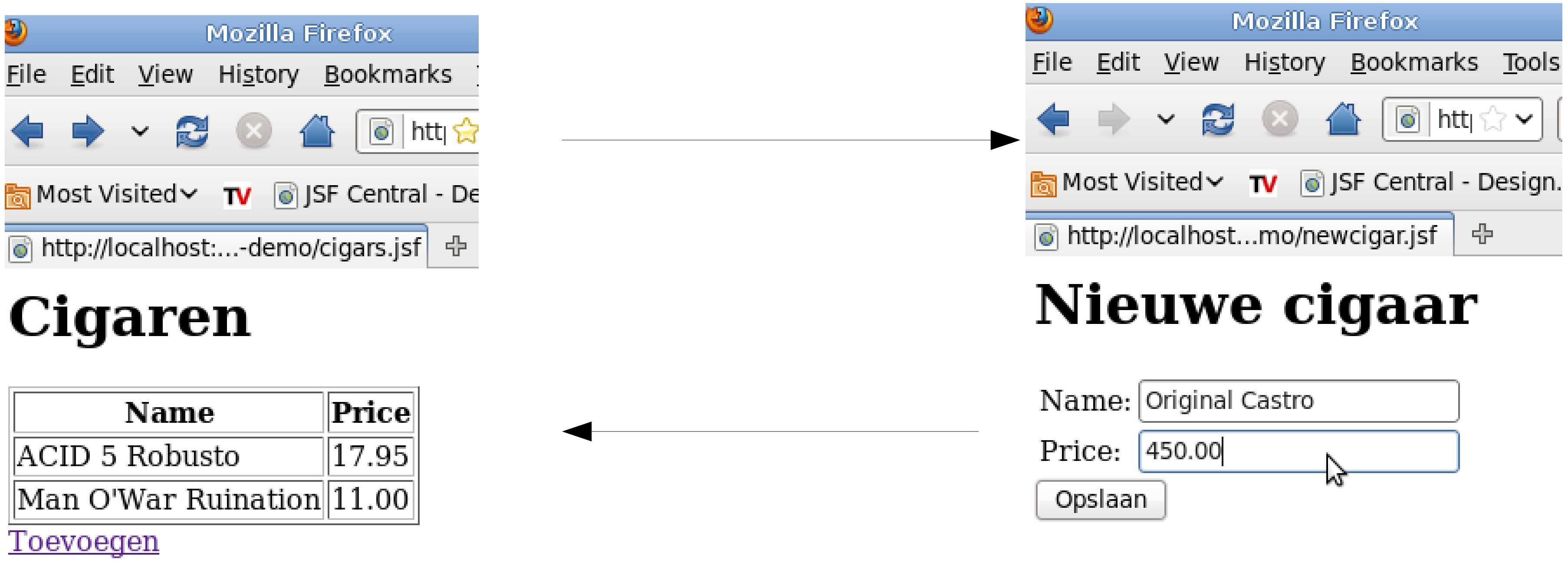
Java EE 6 – Goals and focus

- More ease of development
- Flexible packaging
- Removal of obsolete technologies
- Profiles
    - I.e. web

Java EE 6 – Supporters

- Everyone!
    - Sun/Oracle
    - IBM
    - SpringSource
    - JBoss/Red Hat
    - Apache
    - ...

Java EE 6 – Demo


Mozilla Firefox
File Edit View History Bookmarks
Most Visited TV JSF Central - De
http://localhost:...-demo/cigars.jsf
Mozilla Firefox
File Edit View History Bookmarks Tools
Most Visited TV JSF Central - Design.
http://localhost...mo/newcigar.jsf

Cigaren

Name	Price
ACID 5 Robusto	17.95
Man O'War Ruination	11.00

Toevoegen

Nieuwe cigaar

Name:	Original Castro
Price:	450.00

Opslaan

Demo

eclipselink

Weld

HIBERNATE Validator

github SOCIAL CODING

http://github.com/martijnblankestijn/cigar-shop-demo

Contexts and Dependency Injection

CONSULTING | ICT | OUTSOURCING

ORDINA.OPGELOST.

CDI - Introduction

a powerful set of complementary services that help improve the structure of application code.

CDI - Features

- Typesafe Dependency Injection
- Contexts for stateful objects
- Interceptors & decorators
- Events
- Java EE module and Java EE component support
- Expression Language integration

CDI - Example

```
public class SomeBean {
    @Inject private OtherBean other;

    public void doIt() {
        other.doIt();
    }
```

```
public class OtherBean {
    public void doIt() {
        System.out.println("Doing it");
    }
}
```

CDI – What's a bean?

- No configuration in previous example, any class can be a CDI bean
- So what defines a bean?
  - A set of bean types
  - A set of qualifiers
  - A scope
  - An optional EL name
  - A set of interceptor bindings
  - A bean implementation

CDI – Dependency Injection Example

```
public class HelloServiceImpl implements HelloService {
    public void sayHello(String name) {
        System.out.println("Hello " + name);
    }
}
```

```
@RequestScoped
public class HelloManagedBean {
    @Inject private HelloService svc;

    private String name;

    public void sayHello() {
        svc.sayHello(this.name);
    }
}
```

Bean types:
java.lang.Object
HelloService
HelloServiceImpl

@Any @Default

DependentScoped

CDI – Dependency Injection Example (continued)

```
public class HelloServiceImpl implements HelloService {
    public void sayHello(String name) {
        System.out.println("Hello " + name);
    }
}
```

```
@Named("hello")
@RequestScoped
public class HelloManagedBean {
    @Inject private HelloService svc;

    private String name;

    public void sayHello() {
        svc.sayHello(this.name);
    }
}
```

Bean types:
java.lang.Object
HelloManagedBean

@Any @Default
@Named("hello")

RequestScoped
EL name: hello

CDI - Scope

Every bean has a scope

- Built-in scopes
  - RequestScoped, SessionScoped, ApplicationScoped
    - Standard Servlet scopes
  - ConversationScoped
  - Dependent

- Custom scopes possible
  - Like business process (Seam) or ViewScoped (JSF 2.0)

CDI – Example Named

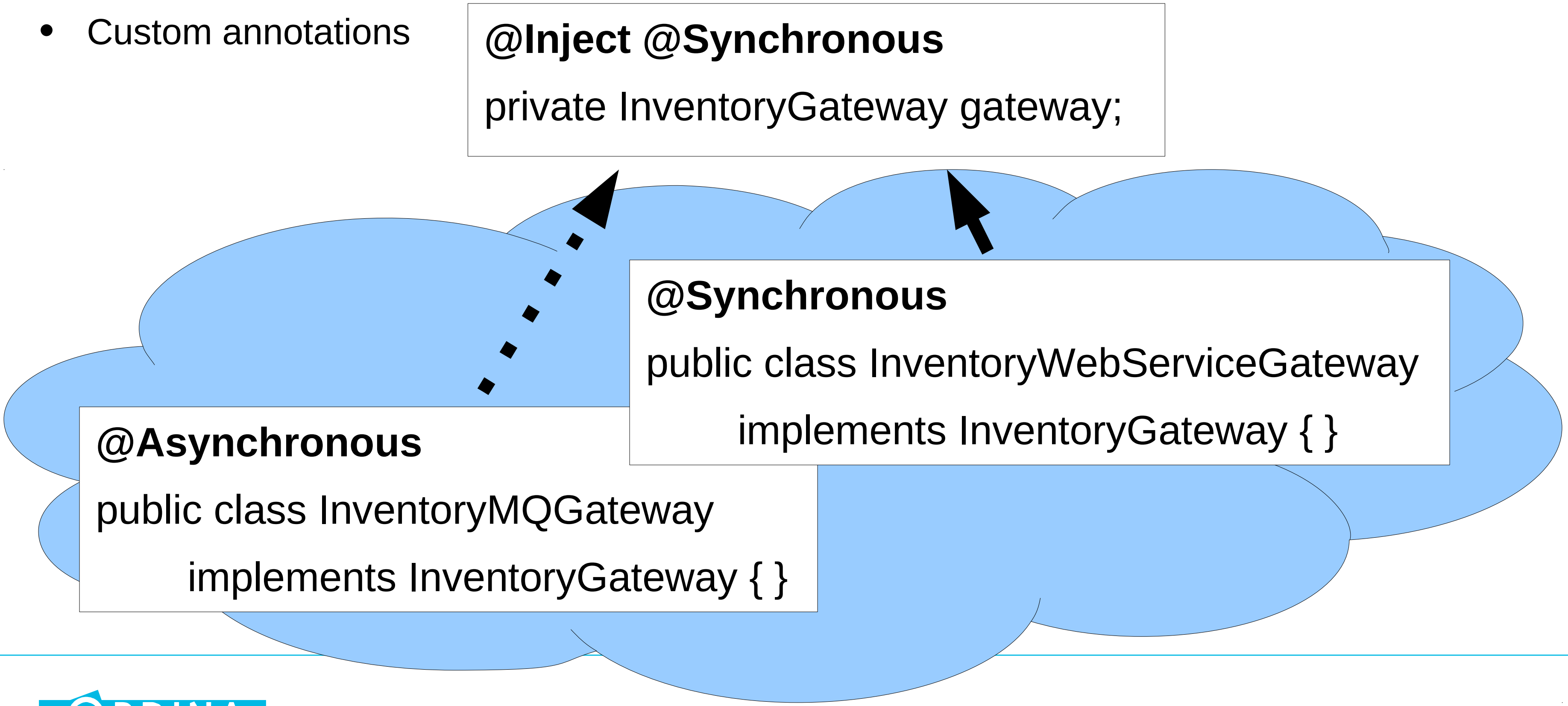
```
<h:dataTable value="#{searchCatalog.products}" var="product">
   <h:column>
       #{product.name}
   </h:column>
</h:dataTable>
```

```
@Named
@RequestScoped
public class SearchCatalog {
    public List<Product> getProducts() { ... }
}
```

CDI – Qualifier

Used to fine tune injection

- If multiple matches, pick the appropriate bean
- Strongly typed
- Custom annotations



```
@Inject @Synchronous
private InventoryGateway gateway;
```

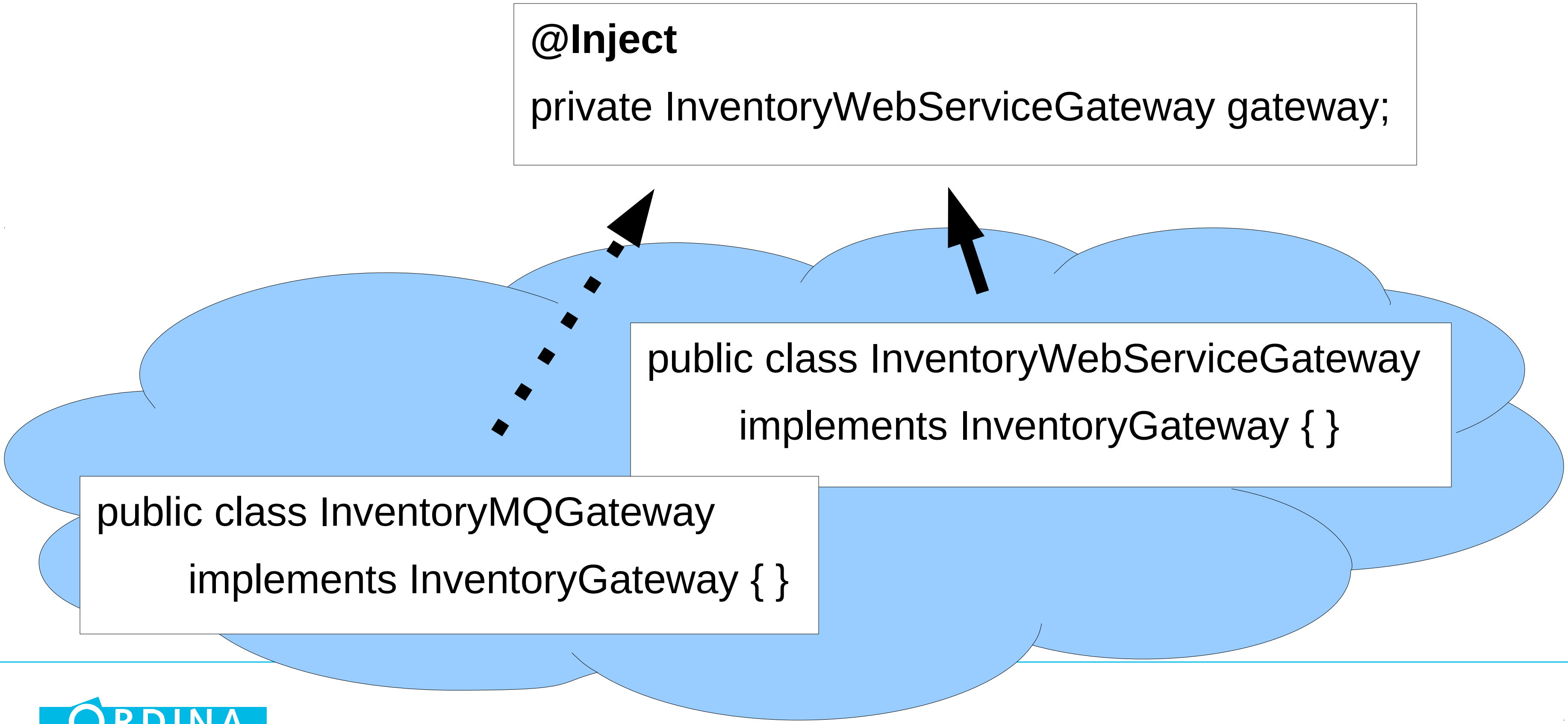
```
@Synchronous
public class InventoryWebServiceGateway
        implements InventoryGateway { }
```

```
@Asynchronous
public class InventoryMQGateway
        implements InventoryGateway { }
```

CDI – Qualifier

You don't have to use qualifiers!

- Less code,
- but also less abstract



```
@Inject
private InventoryWebServiceGateway gateway;
```

```
public class InventoryWebServiceGateway
        implements InventoryGateway { }
```

```
public class InventoryMQGateway
        implements InventoryGateway { }
```

CDI – Stereotype

Abstract multiple concepts into one higher level concept

- @Model is a default stereotype
- Useful for defining architectural concepts

```
@Model
public class ShoppingCartManagedBean {
}
```

```
@javax.inject.Named
@javax.enterprise.context.RequestScoped
@javax.enterprise.inject.Stereotype
public static @interface Model { }
```

CDI – Alternatives

A bean which is always the favorite bean for injection

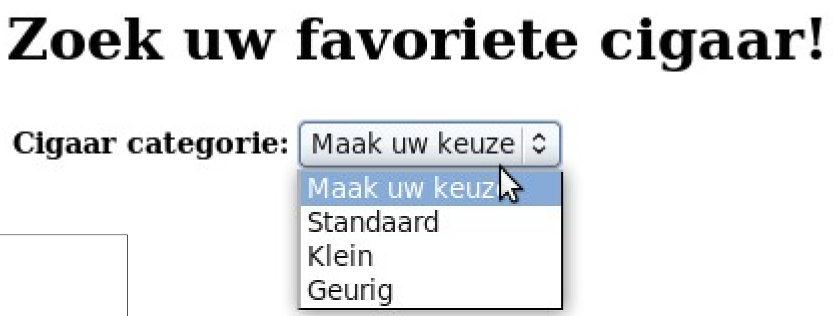
- Very useful for mock implementations in unit tests
- Must be declared in beans.xml
  - Which is a good thing (environment specific stuff should stay out of Java)

```
@Alternative
public class MockEntityManager implements EntityManager { }
```

```
@javax.enterprise.inject.Model
public class MyPaymentController {
    @Inject EntityManager em;
}
```

Only use the mock when testing

CDI – Producer methods


Zoek uw favoriete cigaar!

Cigaar categorie: Maak uw keuze

```
<h:selectOneMenu
        value="#{searchCatalog.currentCategory}" >
    <f:selectItems  value="#{categories}" />
</h:selectOneMenu>
```

```
public class SomeClass {
    @Produces @Named("categories")
    public SelectItem[] getCategories() {
        ...
    }
}
```

CDI – Producer methods

```
@Model
public class OrderManagedBean {
    @Inject FacesContext fc;

    public String doSomething() {
        // Use fc variable
        return null;
    }
}
```

```
public class FacesProducer {
    @Produces @RequestScoped
    public FacesContext getFacesContext() {
        return FacesContext.getCurrentInstance();
    }
}
```

CDI – Events

Can be used for decoupling logic

- Not asynchronous, happens in same thread

```
@Model
public class LoginBean {
    @Inject
    private Event<LoginEvent> event;

    public String login() {
        event.fire(new LoginEvent());
    }
}
```

```
public class MyEventHandler {
    public void loggedIn(
            @Observes LoginEvent event) {
        //do some stuff...
    }
}
```

CDI – Decorator

Implements 1 or more bean types

Intercepts invocations for beans that implement that type

```
public interface CigarService {
    List<Cigar> getCigars();
}
```

```
@Decorator
public class Decorater implements CigarService {
    private final CigarService delegate;

    @Inject
    public Decorater(@Delegate CigarService srv) {
        this.delegate = srv;
    }

    public List<Cigar> getCigars() {
        System.out.println("Before getting cigars");
        return delegate.getCigars();
    }
```

```
<decorators>
   <class>demo.Decorater</class>
</decorators>
```

CDI – Example ..... Too far ?

```
@SessionScoped @Named("login")
public class LoginBean {
    @Inject
    private Event<LoginEvent> event;

    @SuppressWarnings("unchecked")
    public String login() {
        event.fire(new LoginEvent());
    }

    @Produces @ApplicationScoped @MyCustomQualifier
    public Object createSomeObject() { ... }
}
```

JSF 2.0 – There's much more!

- Resource loading and relocation
- ProjectStage
- Exception Handling
- System Events
- Behaviors

Java Server Faces 2.0

JSF 2.0 – What's new

- Ease of development
  - Servlet 3.0
  - Navigation
  - Annotations
  - Debugging
- New scopes
- Facelets included
- Ajax included
- Easy components
- Better error handling
- GET support

JSF 2.0 – Ease of development

Servlet 3.0 support

- Automatically discovery of Faces Servlet
    - Not mentioned in specification
    - But implemented in both Mojarra and MyFaces

    - Zero configuration
        - Faces Servlet mapping: *.jsf and *.faces

    - An empty /WEB-INF/faces-config.xml is enough

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.0" />
```

JSF 2.0 – Navigation

- Navigation example
  - No more Navigation Rules
  - Redirects are also supported
  - And more...

```
public String processOrder() {
    // ...
    return "order_confirmed?faces-redirect=true";
}
```

if there is a page named order_confirmed.xhtml, go there

JSF – Annotations

- Annotations for major JSF artifacts
  - @ManagedBean
    - @ManagedProperty
    - @RequestScoped, @SessionScoped, ApplicationScoped
  - @FacesValidator + @FacesConverter
  - @FacesComponent
  - @FacesRenderer
  - @FacesBehavior

JSF – Managed Beans versus CDI Beans

- There are multiple types of Managed Beans
  - javax.faces.bean.ManagedBean
  - javax.annotation.ManagedBean
- Reasons
  - JSF 2.0 spec finished before CDI spec
  - JSF annotations out-of-the-box
- Good example of bad API design
  - Very annoying for code completion

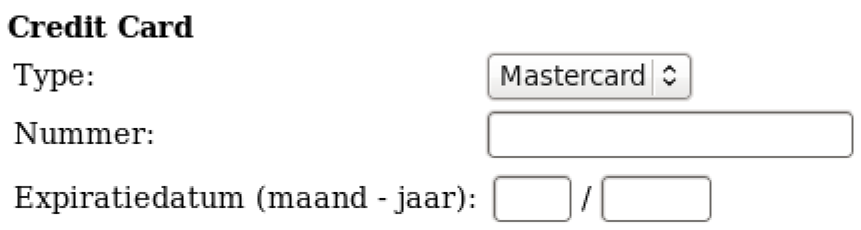
- Workarounds/fixes
  - Startup warnings in JSF applications with CDI on classpath???
  - Always use @Named for Managed Beans

JSF 2.0 – Bookmarkable URLs

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core">
<head>
    <f:metadata>
        <f:viewParam name="id" value="#{myManagedBean.id}"/>
        <f:event type="preRenderView" listener="#{myManagedBean.initialize}"/>
    </f:metadata>
    ...
```

Callback hook for “page load”

JSF 2.0 – Composite User-interface Components



Credit Card

Type:	Mastercard
Nummer:	
Expiratiedatum (maand - jaar):	/

Easy reuse of arbitrary markup

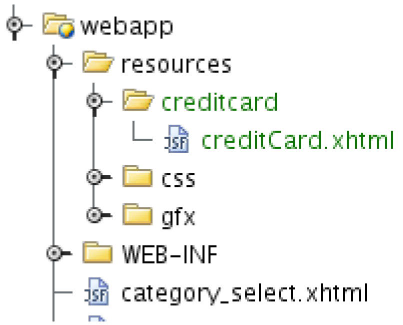
Including validators, converters, etc.

- As simple as possible
- No or minimal configuration
- Support for iterative development

- Already in Facelets, but now even more powerful

JSF 2.0 – Composite Components

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:composition xmlns:cc="http://java.sun.com/jsf/composite/creditcard">
   <cc:creditCard/>

  <html xmlns="http://www.w3.org/1999/xhtml">

  <h:body>

      <composite:interface>

      </composite:interface>

      <composite:implementation>

          <h:panelGrid columns="3" title="#{m.group_credit_card}">

              <h:selectOneMenu id="creditCardType" required="true" value="#{creditCardFormBean.type}">

                  <f:selectItems value="#{creditCardTypes}" />

              </h:selectOneMenu>

          </h:panelGrid>

      </composite:implementation>
```



```
webapp
  resources
    creditcard
      creditCard.xhtml
    css
    gfx
  WEB-INF
  category_select.xhtml
```

JSF 2.0 – AJAX

Interoperable platform

- jsf.js with generic JavaScript
    - Like AJAX stuff

- Also server side
    - Partial state saving
    - Partial response writing

- Targeted at component authors

- Should give more interoperability between component libraries

JSF 2.0 – Scopes

- View Scope
  - Scope which is kept alive as long as the user stays on the same page

- Flash Scope
  - Used to carry data over redirects
  - Main use case: Messages

JSF 2.0 – There's more – Expression Language

```
<h:dataTable value="#{productBean.products}" var="product">
    <h:column>
        <f:facet name="header">Product</f:facet>
        <h:commandLink
            action="#{productBean.select(product.id)}"
            value="#{product.name}" />
    </h:column>
</h:dataTable>
```

Unified Expression Language

JSF 2.0 – There's much more!

- Resource loading and relocation
- ProjectStage
- Exception Handling
- System Events
- Behaviors

Bean Validation

Bean Validation - Introduction

- Metamodel and API for JavaBean validation
- Metadata expressed in
  - Annotations
  - XML validation descriptors

- Across tiers
  - Web, persistence …
  - Server-side, client-side (Swing)

- Bean Validation is actually a Java SE spec
  - ...but integrates with EE technologies

Bean Validation - Usage

```
public class Customer {
    @NotNull
    @Size(min = 2, max = 40)
    private String lastName;
}
```

Bean Validation – Composing annotations

```
@Target({METHOD, FIELD, ANNOTATION_TYPE, …})
@Retention(RUNTIME)
@Constraint(validatedBy = {})
@Min(value = 1) @Max(value = 12)
@NotNull
public @interface Month { }
```

```
public class CreditCardFormBean {

    @Month
    private Integer expiryMonth;
}
```

Bean Validation – Custom Validator

```
@Target({...}) @Retention(RUNTIME)
@Constraint(validatedBy = Impl.class)
public @interface CreditCardYear { }
```

```
public class CreditCardFormBean {

    @CreditCardYear
    private int expiryYear;
}
```

```
public class Impl implements ConstraintValidator<CreditCardYear, Integer> {

    public void initialize(CreditCardYear ccy) { }

    public boolean isValid(Integer year, ConstraintValidatorContext ctx) {
        int now = Calendar.getInstance().get(Calendar.YEAR);
        return year != null && year >= now && year < now + 10;
}
```

Bean Validation – Parametrized Validator

```
@Target({...}) @Retention(RUNTIME)
@Constraint(validatedBy = Impl.class)
public @interface CreditCardYear {
    int expiry() default 10;
}
```

```
public class CreditCardFormBean {

    @CreditCardYear(expiry = 5)
    private int expiryYear;
}
```

```
public class Impl implements ConstraintValidator<CreditCardYear, Integer> {
    private int expiryYear;
    public void initialize(CreditCardYear ccy) {
        expiryYear = ccy.expiry();
    }

    public boolean isValid(Integer year, ConstraintValidatorContext ctx) {
        int now = Calendar.getInstance().get(Calendar.YEAR);
        return year != null && year >= now && year < now + expiryYear;
}
```

Bean Validation – One more thing

Validation Groups

- Define partial validations on types
- I.e. a logged in user doesn't have to provide personal fields

```
public interface KnownCustomer { }
public interface UnknownCustomer { }

public class Payment {
    @NotEmpty private BigDecimal amount;

    @NotNull(groups=KnownCustomer.class) Address address;

    @NotEmpty(groups=UnknownCustomer.class) String street;
    // Other address properties
}
```

Bean Validation – JSF 2.0 integration

JSF BeanValidator

- Automatically uses Bean Validation in JSF classes
  - Plug & Play

- With JSF 2.0 and ClientBehaviors
  - Maybe even validations in JavaScript?

Bean Validation – JPA 2.0 integration

- Automatic validation on JPA 2.0 events
  - pre-persist
  - pre-update
  - pre-remove NOT
- Plug & Play integration
- Generation of DDL from constraints

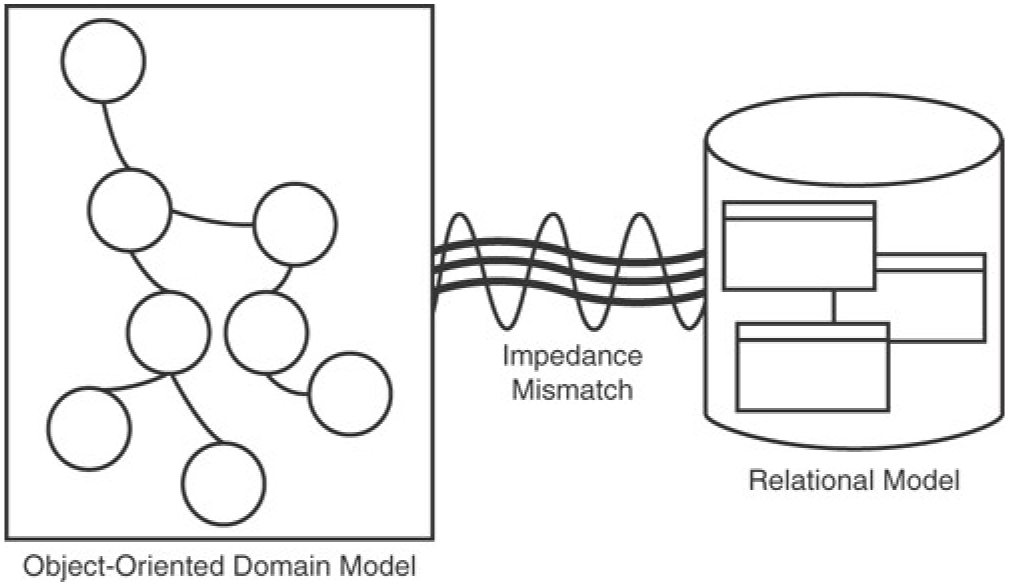
Java Persistence API 2.0

CONSULTING | ICT | OUTSOURCING

ORDINA.OPGELOST.

JPA - Introduction

- API voor management van persistentie en object-relationele mapping voor Java SE en Java EE



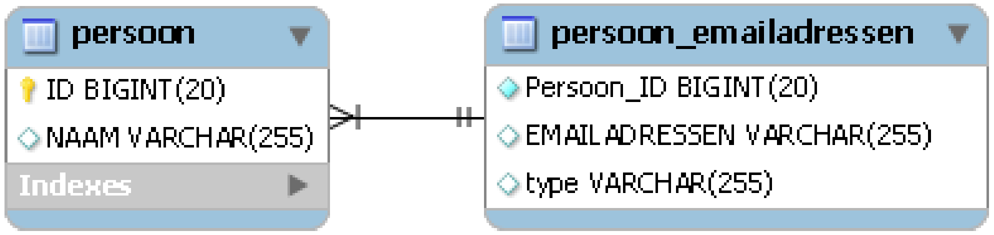
Oracle Kodo™ 4.1

JPA – New in 2.0

- 2.0 een losse specificatie
- Meer mapping mogelijkheden
  - Embeddables
  - Element collections
  - Ordered lists
  - Orphan removal
- Locking
- Uitbreiding Java Persistence Query Language
- MetaModel en API voor Criteria queries
- Type safety
- Cache interface

Element Collection

```
@Entity
public class Persoon {

  @ElementCollection
  @MapKeyColumn(name = "type")
  private Map<String, String> emailadresssen;


  public void addEmailadres(String type, String emailadres) {
       emailadresssen.put(type, emailadres);
  }
```



- Je hebt uitgebreide mogelijkheden om het type van de Key en Value te bepalen:
  - Embeddable, Basic, Entity

JPA – JPQL

In JPA 1.0 alleen de Java Persistence Query Language (JPQL)

```
Query<Docent> q =
        em.createQuery("select d from Docent d", Docent.class);
List<Docent> list = q.getResultList();
```

JPA - Criteria API

- Typesafe object-based Query API op basis van het Metamodel van Entiteiten
- `CriteriaQuery<Object> createQuery()`

- Bijvoorbeeld:

```
CriteriaQuery<Docent> q =
             em.getCriteriaBuilder().createQuery(Docent.class);
List<Docent> docenten = em.createQuery(q).getResultList();
```

Criteria API – Selectief data ophalen

- QueryBuilder bevat alle operaties (equal, like, gt …)

```
CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Persoon> q =  cb.createQuery(Persoon.class);
Root<Persoon> from = q.from(Persoon.class);

q.where(cb.equal(from.<String>get("achternaam"), "Smit"));
```

Criteria API – MetaModel API

- Nadeel string based property access: runtime controle op juistheid attribuutnaam
- Het MetaModel brengt de oplossing!

```
Predicate la = cb.like(from.get(Persoon_.achternaam), "Sm%t");
Predicate lv = cb.like(from.get(Persoon_.voornaam), "Martin");
criteriaQuery.where(cb.and(la, lv));
```

- De mysterieuze Persoon_ klasse bevat de attributen van Persoon met hun type.
- De JPA genereert deze.

```
@StaticMetamodel(Persoon.class)
public class Persoon_ {

public static volatile SingularAttribute<Persoon, String> achternaam;
public static volatile SingularAttribute<Persoon, String> voornaam;
```

Enterprise JavaBeans 3.1

CONSULTING | ICT | OUTSOURCING

ORDINA.OPGELOST.

Wat is er nieuw?

- Interface-less EJB’s
- Singletons
- Scheduling
- Standard Global JNDI names
- EJB’s in de webapplicatie (EJB Lite) *
- Asynchrone aanroep *
- Embedded Container *

EJB 3.1 – Lite

subset voor het schrijven van **portable, transactionele** business logica

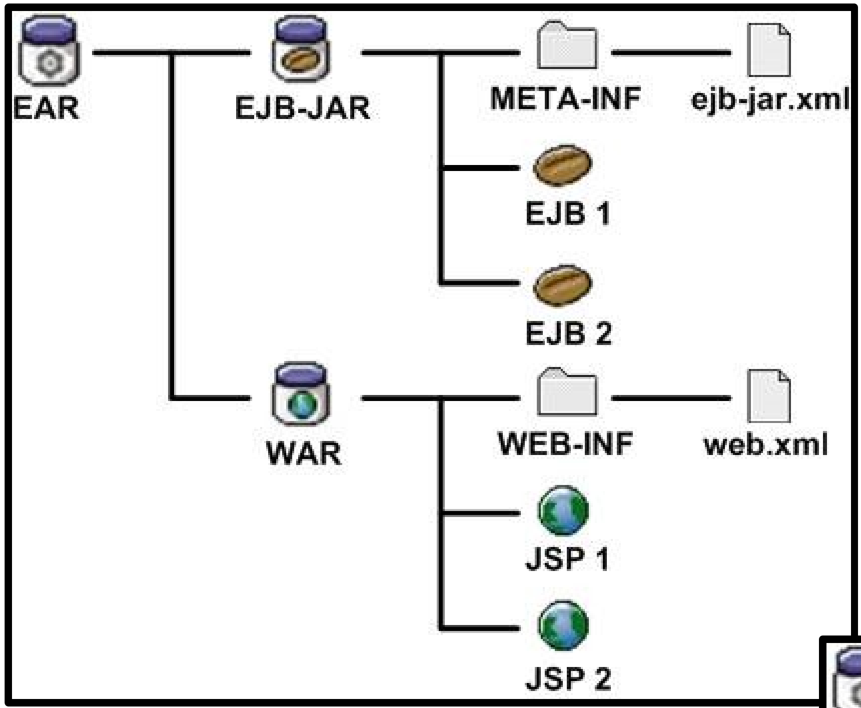
Je hebt:

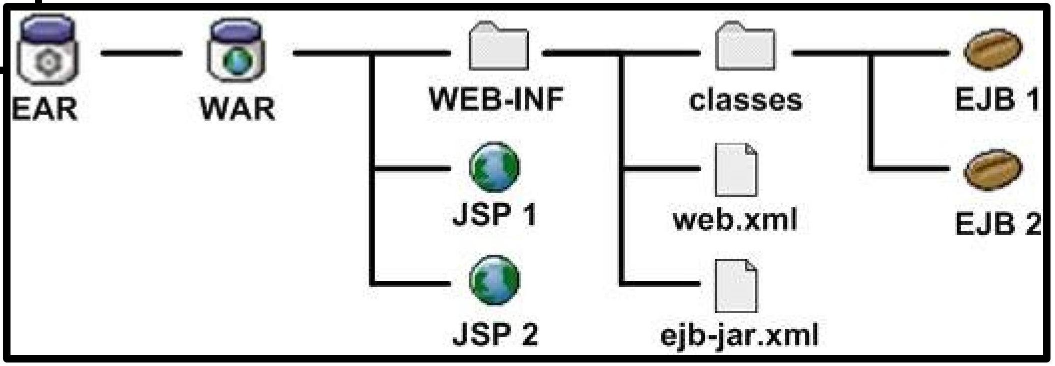
- Stateful, stateless, en singleton session beans
- JPA
- Local / No-interface
- Interceptors
- Container/Bean managed transactions
- Declarative security

Restricties:

Je hebt geen:
- Message-Driven Beans
- Remoting
- JAX-WS WebService endpoint
- EJB Timer Service
- Asynchronous Invocation

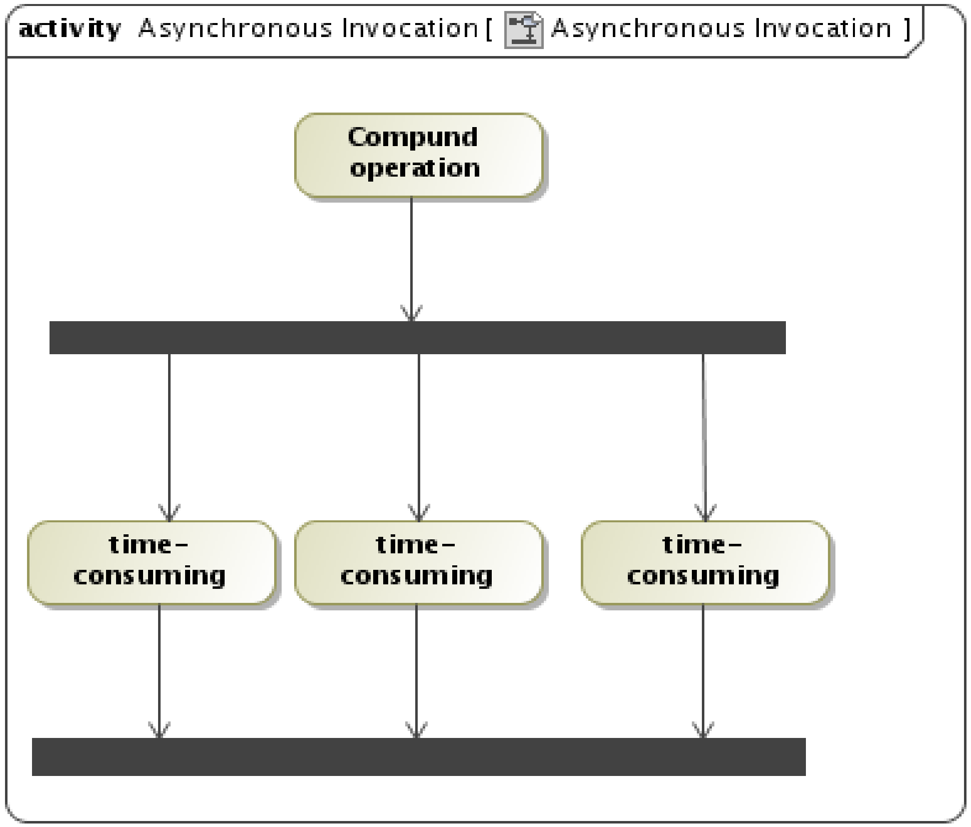
EJB 3.1 – Lite


EAR
EJB-JAR
META-INF
ejb-jar.xml
EJB 1
EJB 2
WAR
WEB-INF
web.xml
JSP 1
JSP 2


EAR
WAR
WEB-INF
classes
EJB 1
JSP 1
web.xml
EJB 2
JSP 2
ejb-jar.xml

EJB 3.1 – Asynchronous Invocation

Asynchrone aanroep van stateless session beans


activity Asynchronous Invocation [Asynchronous Invocation]
Compund operation
time-consuming
time-consuming
time-consuming

EJB 3.1 – Asynchronous Invocation Example

```
@Asynchronous
public Future<String> asynchronousHello(String label) {
        Thread.sleep(5000);
    return new AsyncResult<String>("Hello " + label);
}
```

```
Future<String> future= bean.asynchronousHello("World");

while(result == null) {
    try {
        result = future.get(1, TimeUnit.SECONDS);
    } catch (TimeoutException e) {
        System.out.println("Waiting 1 ....");
    }
}
System.out.println("Got result: " + result);
```

EJB 3.1 – Embedded Container

Uitvoering van EJB applications in een Java SE-omgeving

- Client en EJB-Container in dezelfde JVM
- Zelfde basale services als in een Java EE runtime
  injection, component environment, container-managed transactions
- Alleen support voor EJB-Lite vereist

Voordelen:

- Betere testbaarheid
- Offline (batch) processing
- Gebruik EJB in desktop applicaties

EJB 3.1 – Embedded Container Example

```
final EJBContainer ec = EJBContainer.createEJBContainer();
```

Default:

- Container gebruikt JVM Classpath to scan for EJB's

```
final Context context = ec.getContext();
```

And what more?

Java EE Profiles - Introduction

a configuration of the platform

suited to a particular class of applications

- Subset van technologies ==> dropping non-relevant technologies
- Add technologies ==> Java EE Portal Profile probably includes Portlet API
- Optional technologies

Java EE Profiles – Web Profile

Required

- Servlet. JSP
- JSF, Expression Language
- EJB Lite
- JPA
- JTA
- Bean Validation
- Contexts and Dependency Injection

Java EE - Pruning

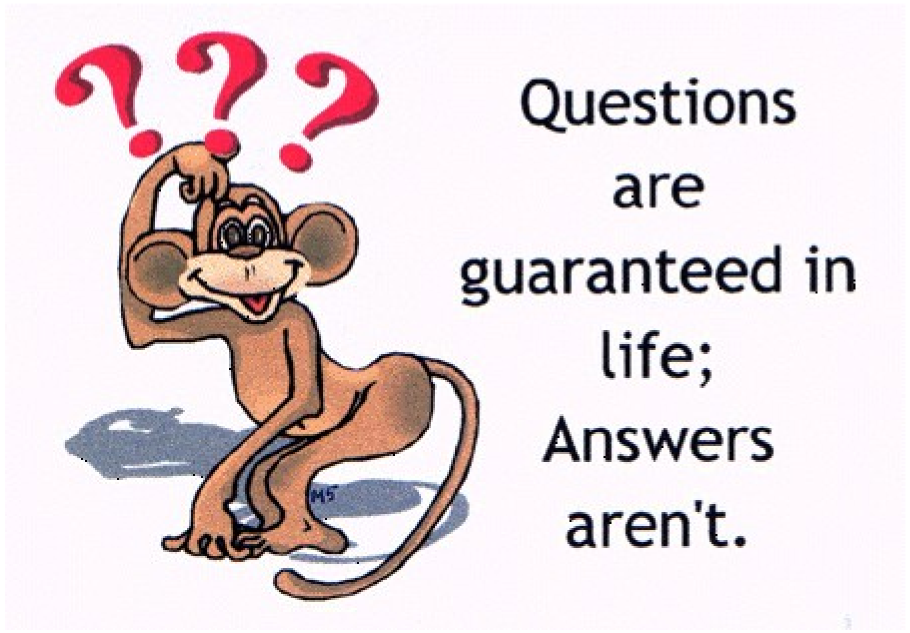
Proces voor verwijderen van technologieen

- Versie N aankondiging
- Versie N + 1 Mogelijk verwijdering

Nu genomineerd:

- Entity Beans
- JAX-RPC
- JAXR

TENSLOTTE



Ordina
Ringwade 1
3439 LM Nieuwegein
Tel. 030 663 .. ..
www.ordina.nl