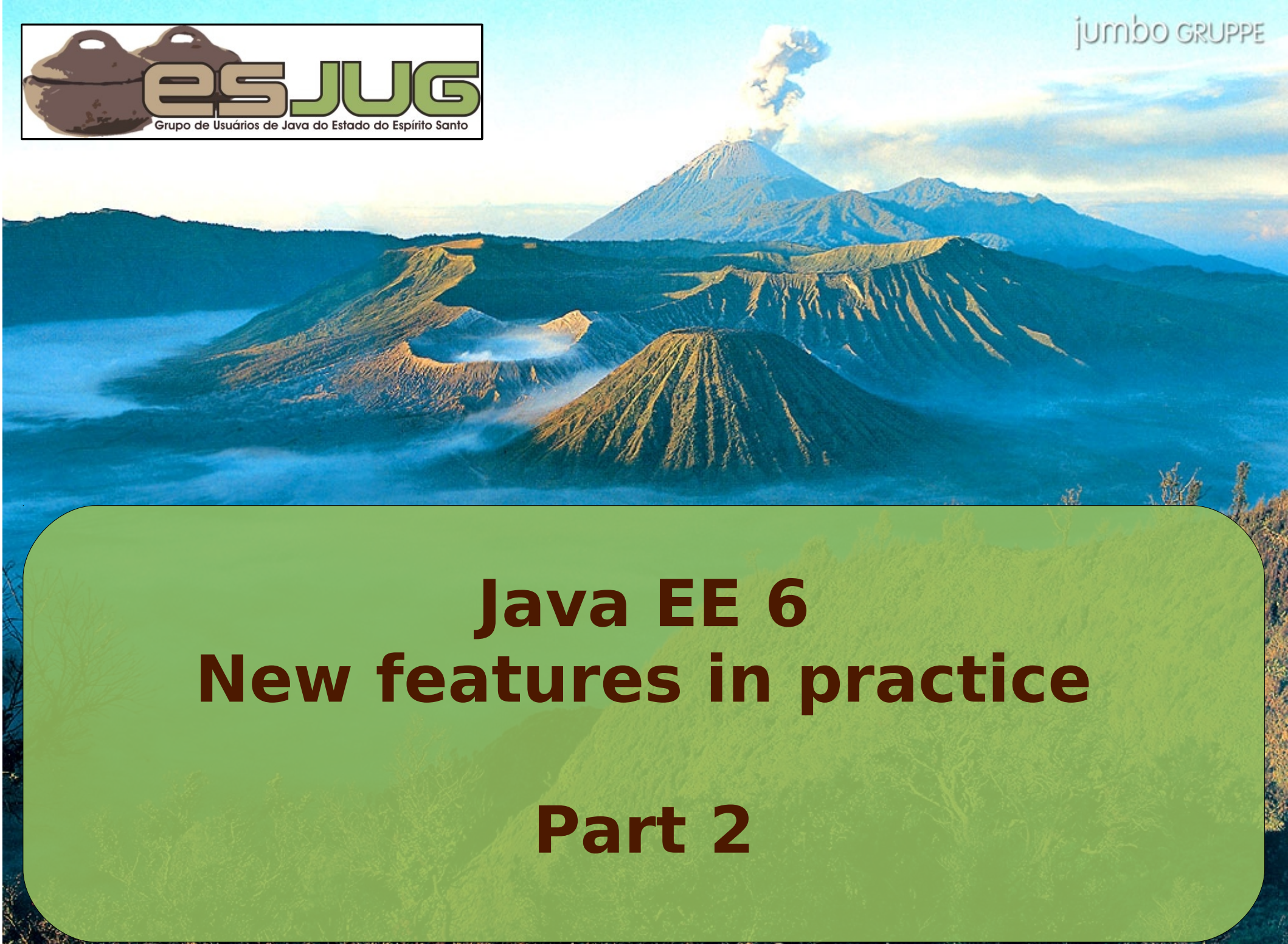


Java EE 6
New features in practice

Part 2

License for use and distribution

This material is available for non-commercial use and can be derived and/or redistributed, as long as it uses an equivalent license.

Attribution-Noncommercial-Share Alike 3.0 Unported

http://creativecommons.org/licenses/by-nc-sa/3.0/

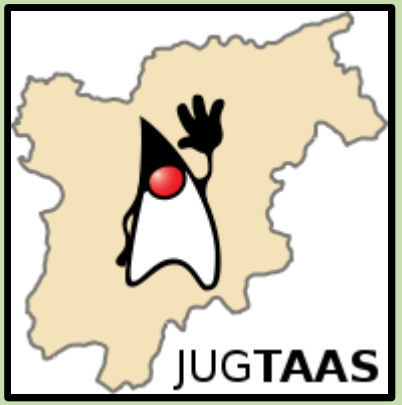
You are free to share and to adapt this work under the following conditions: (a) You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work); (b) You may not use this work for commercial purposes. (c) If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

About the author – Vítor Souza

- Education:
  - Computer Science graduate, masters in Software Engineering – (UFES, Brazil), taking PhD at U. Trento.
- Java:
  - Developer since 1999;
  - Focus on Web Development;
  - Co-founder and coordinator of ESJUG (Brazil).
- Professional:
  - Substitute teacher at Federal University of ES;
  - Engenho de Software Consulting & Development.
- Contact: vitorsouza@gmail.com

JUG TAAS = JUG Trento + JUG Bolzano

- Website:
  - http://www.jugtrento.org/
  - http://www.jugbz.org/
- Mailing list (in Italian, mostly): http://groups.google.com/group/jugtaa
- If you're interested in Java, join and participate!



JUGTRENTO.ORG
Java User Group

Agenda

- Quick summary of part 1;
- Facelets for page decoration;
- Criteria API;
- Conversations;
- AJAX support.

Quick summary of part 1 (1)

- Java EE 6 (JSR 316), released in December 2009;
  - Platform for development of enterprise applications (scalability, security, accessibility, etc.);
  - New version focuses on flexibility, extensibility and ease of development;
- The platform includes many other technologies:

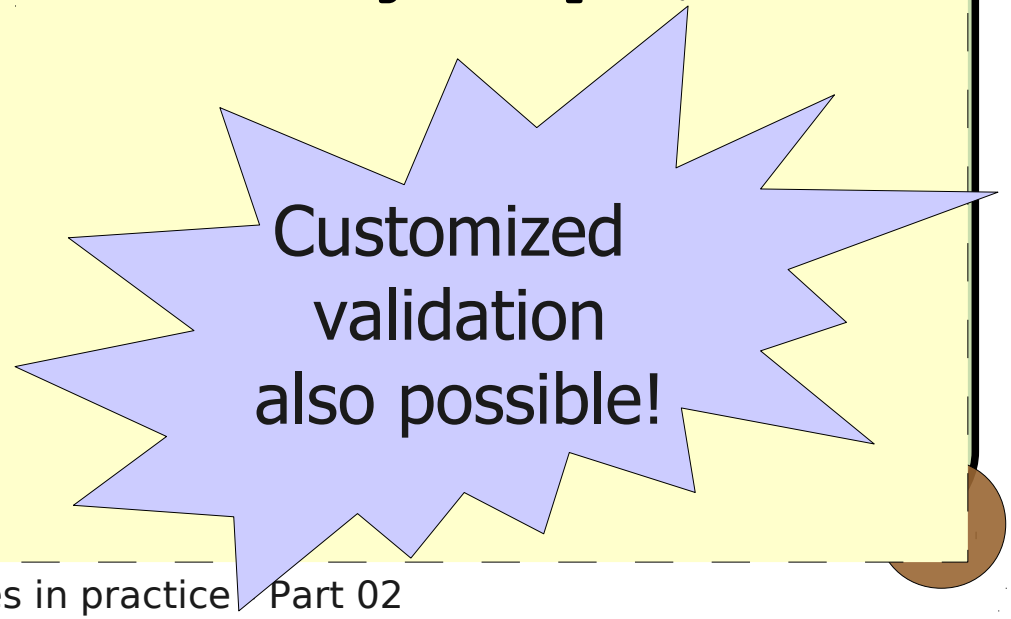
  Bean Validation, CDI, EJB, EL, JACC, JASPIC, Deployment API, Management API, JavaMail, JAX-RS, JAX-WS, JAXB, JCA, JMS, JPA, JSF, JSP, JSTL, JTA, Managed Beans, Servlet, Web Services Metadata.

- Profiles: standard, Web, more in the future?

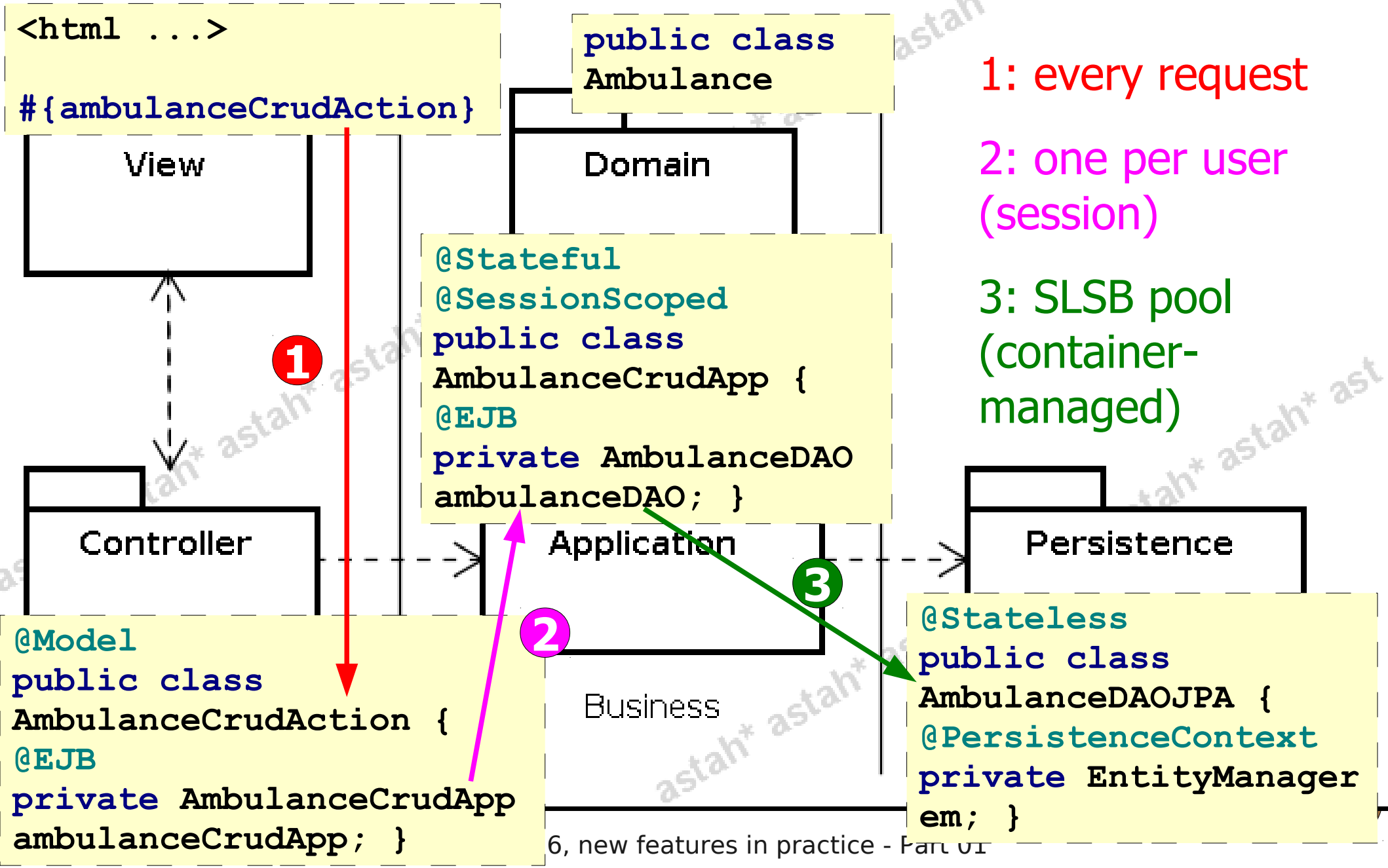
Quick summary of part 1 (2)

- Tools: GlassFish V3, NetBeans 6.9;
- Example application: ADS;
- Domain objects: POJO + JPA Annotations;
- Bean Validation: annotations on domain classes provide validation across the platform.

```
public class Ambulance extends PersistentObjectImpl {
   @NotNull
   private int number;

   @NotNull
   @Size(min = 8, max = 8)
   private String licensePlate;

   /* ... */
}
```



Quick summary of part 1 (3) – CDI



Facelets

Facelets

- Alternative to JSP for JSF pages (since 2005);
- JSF and JSP are incompatible (see [1]);
- Was framework, became standard in Java EE 6;
- Web pages written in XHTML (verifiable);
- Allows the construction of decorators for pages;
- Allows the creation of custom components.

[1] = onjava.com/pub/a/onjava/2004/06/09/jsf.html

Facelets decorators

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE ...>
<html ...>
   <h:head>
      <link href="#{facesContext.externalContext.
         requestContextPath}/arquivos/estilos/style.css"
         rel="stylesheet" type="text/css" media="screen" />
      <title><h:outputText value="ADS :: " />
         <ui:insert name="title" /></title>
   </h:head>
   <h:body>
      <!-- Header... -->
      <ui:insert name="content">Default text</ui:insert>
      <!-- Footer... -->
   </h:body>
</html>
```

Using Facelets decorators

```
<!DOCTYPE ...>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:h="http://java.sun.com/jsf/html"
            template="/resources/templates/decorador.xhtml">
   <ui:define name="title">Welcome</ui:define>

   <ui:define name="content">
      <h1>Welcome to the ADS</h1>

      <p>Some information...</p>
   </ui:define>
</ui:composition>
```

Custom components with Facelets

```
<ui:composition ...>
   <table align="center" border="0" cellpadding="3">
      <ui:insert />
   </table>
</ui:composition>
```

form.xhtml

```
<ui:composition ...>
   <tr>
      <td align="right" valign="top">
         <ui:insert name="fieldName" />:
      </td>
      <td><ui:insert /></td>
   </tr>
</ui:composition>
```

field.xhtml

```
<ui:composition ...>
   <tr>
      <td colspan="2" align="right">
         <ui:insert />
      </td>
   </tr>
</ui:composition>
```

button.xhtml

Using custom components

```
<ui:decorate template="/resources/templates/form.xhtml">
<h:form>
   <ui:decorate template="/resources/templates/field.xhtml">
      <ui:define name="fieldName">Username</ui:define>
      <h:inputText size="15" />
   </ui:decorate>
   <ui:decorate template="/resources/templates/field.xhtml">
      <ui:define name="fieldName">Password</ui:define>
      <h:inputSecret size="15" />
   </ui:decorate>
   <ui:decorate
      template="/resources/templates/button.xhtml">
      <h:commandButton value="Log in" />
   </ui:decorate>
</h:form>
</ui:decorate>
```

Example of a complex component

```
<ui:composition ...>
   <table border="0" class="formField #{(fieldName == null
or empty facesContext.getMessageList(fieldName)) ? '' :
'formFieldError'}">
      <tr>
         <td class="label #{(fieldName == null or empty
facesContext.getMessageList(fieldName)) ? '' :
'labelError'}" valign="top">
            <ui:insert name="label" /><h:panelGroup
styleClass="star" rendered="#{(fieldName != null and
facesContext.viewRoot.findComponent(fieldName).required)}">*
</h:panelGroup>:
         </td>
         <td class="spacing"></td>
         <td class="field #{(fieldName == null or empty
facesContext.getMessageList(fieldName)) ? '' :
'fieldError'}">
            <h:messages for="#{fieldName}" layout="table"
rendered="#{fieldName != null}" />
            <ui:insert />
</td></tr></table></ui:composition>
```

JPA 2.0 Criteria API

Criteria API

- New in JPA 2.0;
- Before there was JPQL only;
- Similar to Hibernate Criteria API (like JPQL is similar to HQL);
- Allows programmatic construction of queries;
  - Uses objects instead of Strings;
  - Thus, can be verified at compile time.
- Two modes: static and dynamic.

Criteria API – Dynamic mode

```
public Employee retrieveByUsername(String username) {
   CriteriaBuilder cb = em.getCriteriaBuilder();
   CriteriaQuery<Employee> cq =
      cb.createQuery(Employee.class);
   Root<Employee> root = cq.from(Employee.class);

   EntityType<Funcionario> model = root.getModel();
   cq.where(cb.equal(root.get(model.getSingularAttribute(
      "login", String.class)), username));

   Funcionario funcionario = null;
   try {
      funcionario = em.createQuery(cq).getSingleResult();
   } catch (RuntimeException e) {
      /* Do something... */
      return null;
   }

   return funcionario;
}
```

Dynamic

Criteria API – Static mode

```
public Employee retrieveByUsername(String username) {
   /* Same stuff before... */

   cq.where(cb.equal(root.get(EmployeeJPAMetamodel.login),
      username));

   /* Same stuff after... */
```

Static

```
package it.unitn.disi.ads.core.persistence;

import it.unitn.disi.ads.core.domain.Employee;
import it.unitn.disi.ads.core.domain.EmployeeType;
import javax.persistence.metamodel.SingularAttribute;
import javax.persistence.metamodel.StaticMetamodel;

@StaticMetamodel(Employee.class)
public class EmployeeJPAMetamodel {
 public static volatile SingularAttribute<Employee, String> name;
 public static volatile SingularAttribute<Employee, String> login;
 public static volatile SingularAttribute<Employee, String>
    password;
 public static volatile SingularAttribute<Employee, EmployeeType>
    type;
}
```

Dynamic x Static

```
EntityType<Funcionario> model = root.getModel();
cq.where(cb.equal(root.get(model.getSingularAttribute(
   "login", String.class)), username));
```

Dynamic

```
cq.where(cb.equal(root.get(EmployeeJPAMetamodel.login),
   username));
```

Static

- Dynamic model uses String (prone to error);
- Static model requires an extra class (meta-model);
  - Code generators could easily help here...

Conversations in CDI

CDI Scopes

- Determine when beans exist and are bound;
- Are extensible: create your own scope;
- Five scopes already provided:
  - Defined by the Servlet API: Request, Session, Application;
  - Dependent scope: the bean's scope is the same as the bean that injected it;
  - Conversation scope: a collection of requests within a session, established programmatically.

Conversations

- Are **transient** by default;
- Transient conversations begin/end with the request;
- Can be programmatically changed to **long-running**:

```
public class ReceiveCallAction {
  @Inject
  private Conversation conversation;

  public void someMethod() {
    if (conversation.isTransient()) conversation.begin();

    /* ... */
  }
}
```

Conversations

- Long-running conversations last until programmatically ended:

```
public class ReceiveCallAction {
  @Inject
  private Conversation conversation;

  public void someOterMethod() {
    /* ... */

    If (! conversation.isTransient()) conversation.end();
  }
}
```

- Objects with conversation scope are bound to that context until the conversation ends.

Conversations

- Propagation:
  - If navigation is done through JSF (e.g. `<h:commandLink />`), propagation is automatic;
  - Otherwise, you can use `?cid=X` in the URL;
- Conversation management:
  - Conversation ids can be changed (give it a name);
  - Conversations can be stored in a collection of a session-scoped bean;
  - User can change the active conversation using the `cid` parameter in the URL.

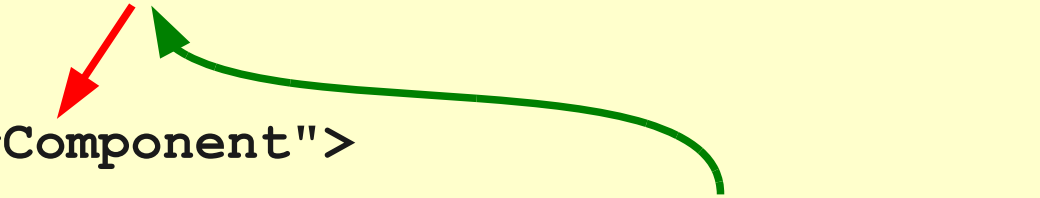
AJAX Support

Support for AJAX in JSF 2.0

- New tag `<f:ajax />`;
- Attributes:
  - `event`: which event should trigger the request (`action`, `blur`, `change`, `click`, ...);
  - `listener`: method to execute when the event occurs;
  - `execute`: data to submit in the request (`@all`, `@none`, `@this`, `@form`, component IDs);
  - `render`: what should be redrawn (`@all`, `@none`, `@this`, `@form`, component IDs).

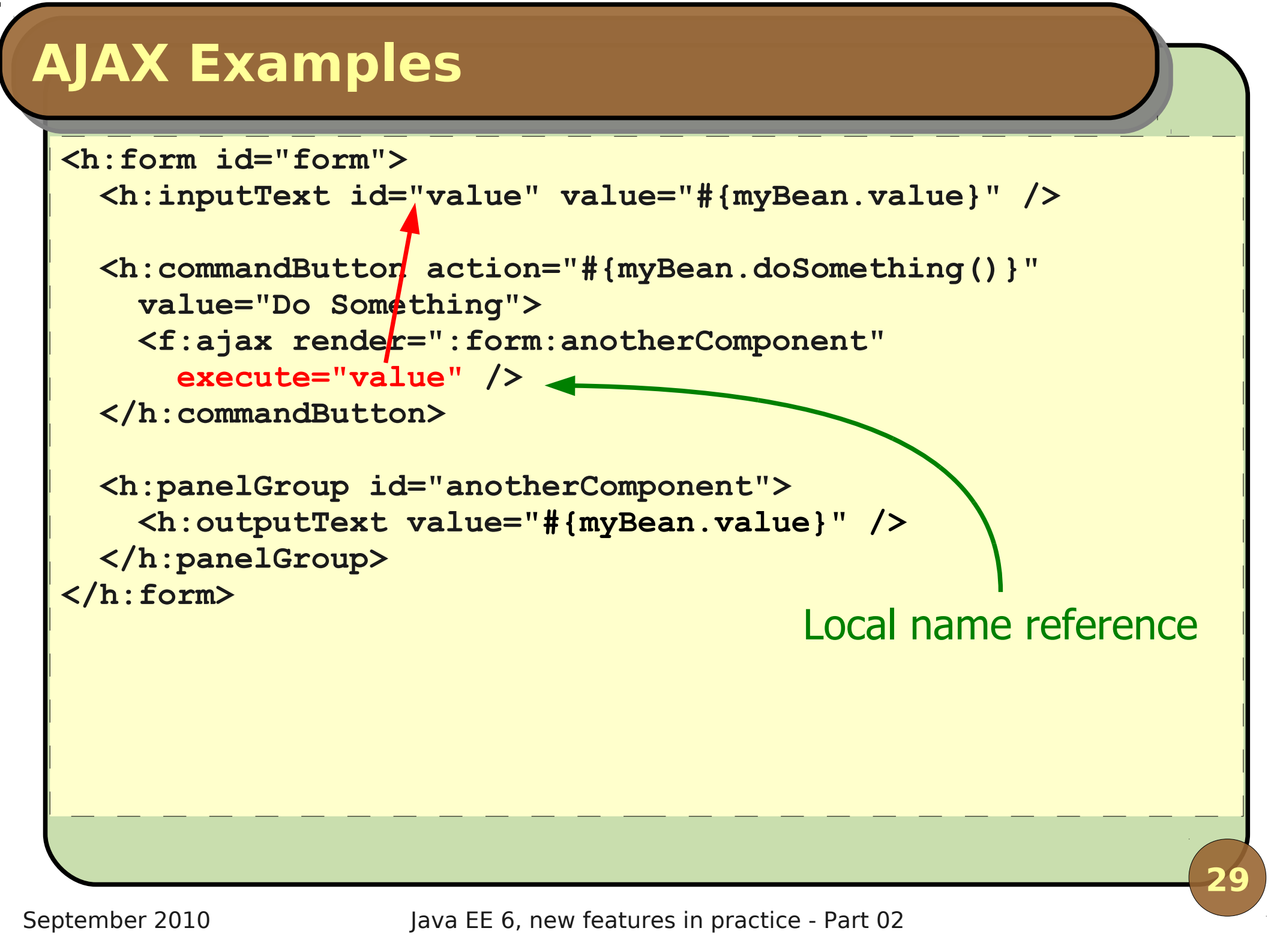
AJAX Examples

```
<h:form id="form">
  <h:commandButton action="#{myBean.doSomething()}"
    value="Do Something">
    <f:ajax render=":form:anotherComponent" />
  </h:commandButton>

  <h:panelGroup id="anotherComponent">
    <!-- ... -->
  </h:panelGroup>
</h:form>
```



AJAX Examples

```
<h:form id="form">
  <h:inputText id="value" value="#{myBean.value}" />

  <h:commandButton action="#{myBean.doSomething()}"
    value="Do Something">
    <f:ajax render=":form:anotherComponent"
      execute="value" />
  </h:commandButton>

  <h:panelGroup id="anotherComponent">
    <h:outputText value="#{myBean.value}" />
  </h:panelGroup>
</h:form>
```



AJAX Examples

```
<h:form id="form">
  <h:inputText id="name" value="#{myBean.obj.name}">
    <f:ajax event="blur" render="acronym"
      listener="#{myBean.suggestAcronym}" />
  </h:inputText>

  <h:inputText id="acronym" value="#{myBean.obj.acronym}" />

  <!-- ... -->
</h:form>
```

```
@Named
public class MyBean {
  private DomainObject obj = new DomainObject();
  public DomainObject getObj() { return obj; }

  public void suggestAcronym(AjaxBehaviorEvent event) {
    String acronym = /* Calculate an acronym. */
    obj.setAcronym(acronym);
  }
}
```

AJAX and Bean Validation

1 - Add validation annotation to your entities;

2 - Use the entity to exchange data with JSF (model-driven);

3 - Use a Facelets custom component to put fields in the form;

4 - Add `<f:ajax />` to form fields, blur event, re-render the whole Facelets component;

5 - In your Facelets component code, check for JSF messages on that field and change color if there are errors.

Remembering Bean Validation

```
public class Ambulance extends PersistentObjectImpl {
   @NotNull
   private int number;

   @NotNull
   @Size(min = 8, max = 8)
   private String licensePlate;

   /* ... */
}
```

Remembering Facelets components

```
<ui:composition ...>
   <table border="0" class="formField #{(fieldName == null
or empty facesContext.getMessageList(fieldName)) ? '' :
'formFieldError'}">
      <tr>
         <td class="label #{(fieldName == null or empty
facesContext.getMessageList(fieldName)) ? '' :
'labelError'}" valign="top">
            <ui:insert name="label" /><h:panelGroup
styleClass="star" rendered="#{(fieldName != null and
facesContext.viewRoot.findComponent(fieldName).required)}">*
</h:panelGroup>:
         </td>
         <td class="spacing"></td>
         <td class="field #{(fieldName == null or empty
facesContext.getMessageList(fieldName)) ? '' :
'fieldError'}">
            <h:messages for="#{fieldName}" layout="table"
rendered="#{fieldName != null}" />
            <ui:insert />
</td></tr></table></ui:composition>
```

Finally, the form

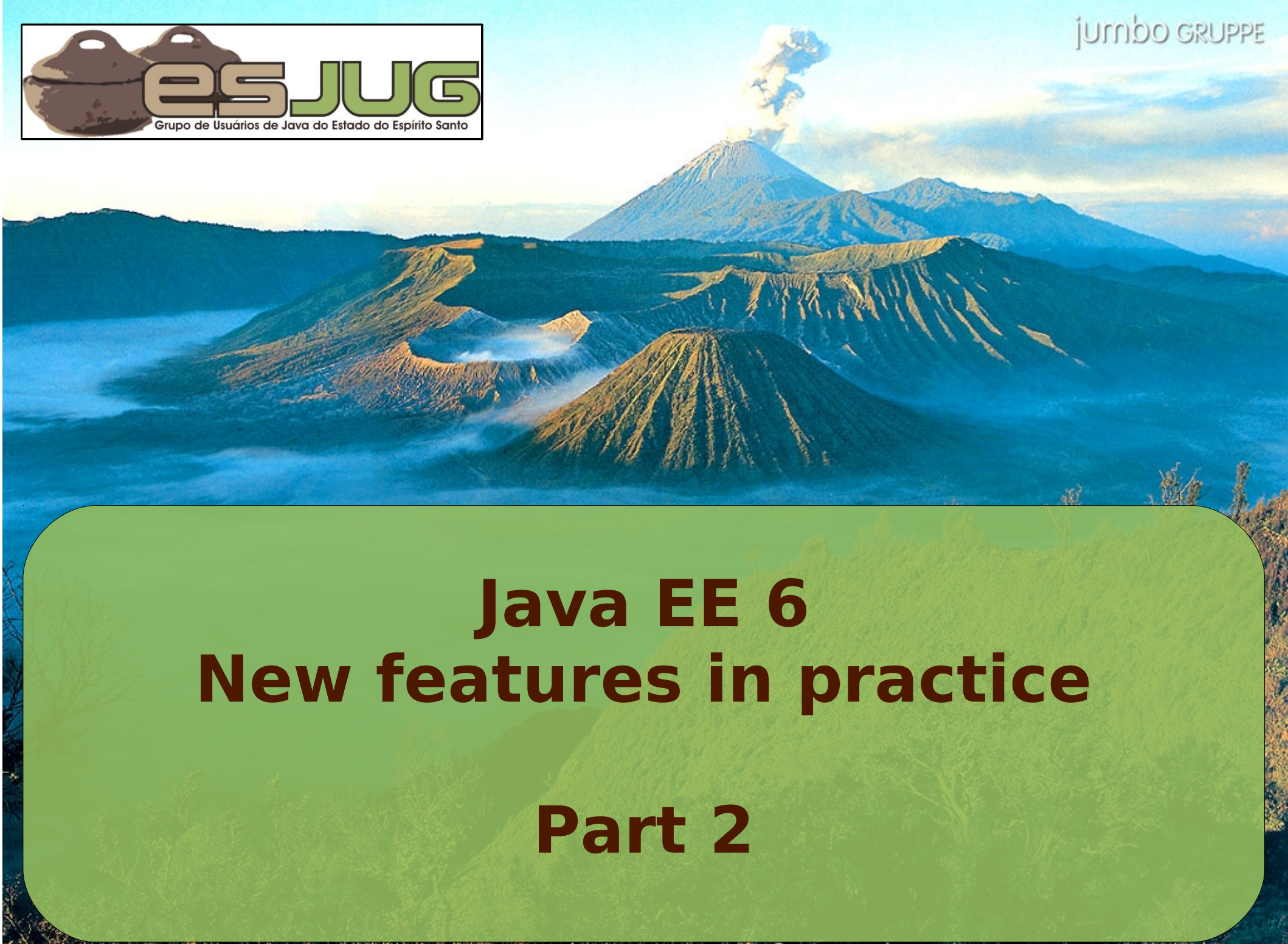
```
<ui:decorate template="/templates/form.xhtml">
  <h:form id="form">
    <h:panelGroup id="numberField">
      <ui:decorate template="/templates/field.xhtml">
        <ui:param name="fieldName" value="form:number" />
        <ui:define name="label">Number</ui:define>
        <h:inputText id="number"
          value="#{ambulanceCrudBean.ambulance.number}">
          <f:ajax event="blur" render="numberField" />
        </h:inputText>
    </ui:decorate>
  </h:panelGroup>

  <!-- ... -->

  </h:form>
</ui:decorate>
```

Conclusions

- Facelets can provide decorator templates and custom components for forms and etc.;
- The Criteria API allows us to check our queries at compile time in exchange for increased complexity in the code;
- Conversations provide a new scope to which objects can be bound, also allowing for management of multiple conversations;
- JSF 2.0 comes with built-in AJAX support.

Would you like to know more?

- Part 3, if you're interested, talks about:
  - JAAS, the Java Authentication and Authorization Services – manage users in the Application Server;
  - Servlets 3.0: what's new in the oldest specification of Java for the Web;
  - More JPA 2.0: new JPQL commands;
  - EJB enhancements: no-interface EJBs, singleton EJBs, asynchronous methods.



Java EE 6
New features in practice

Part 2