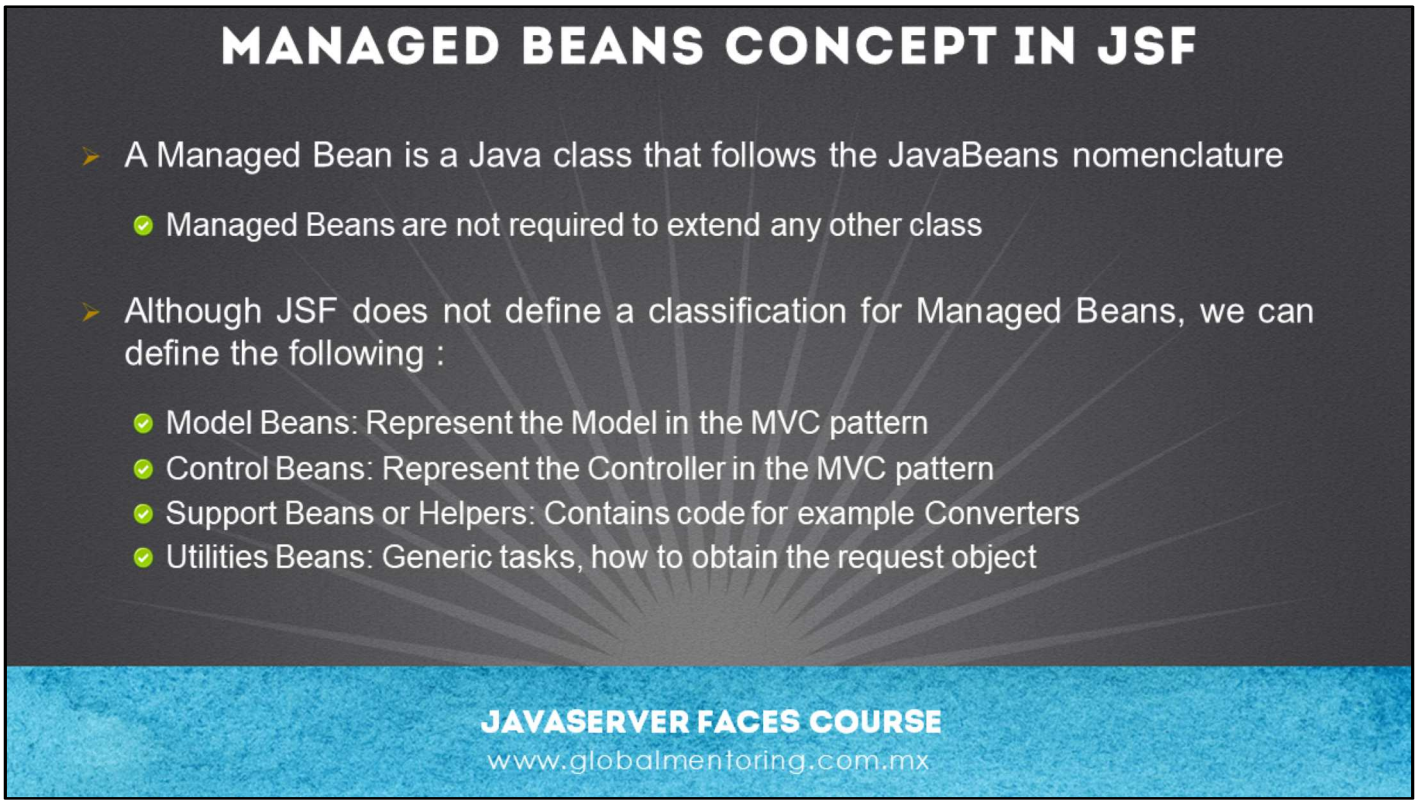

Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the concept of Managed Beans in JSF.

Are you ready? Come on!



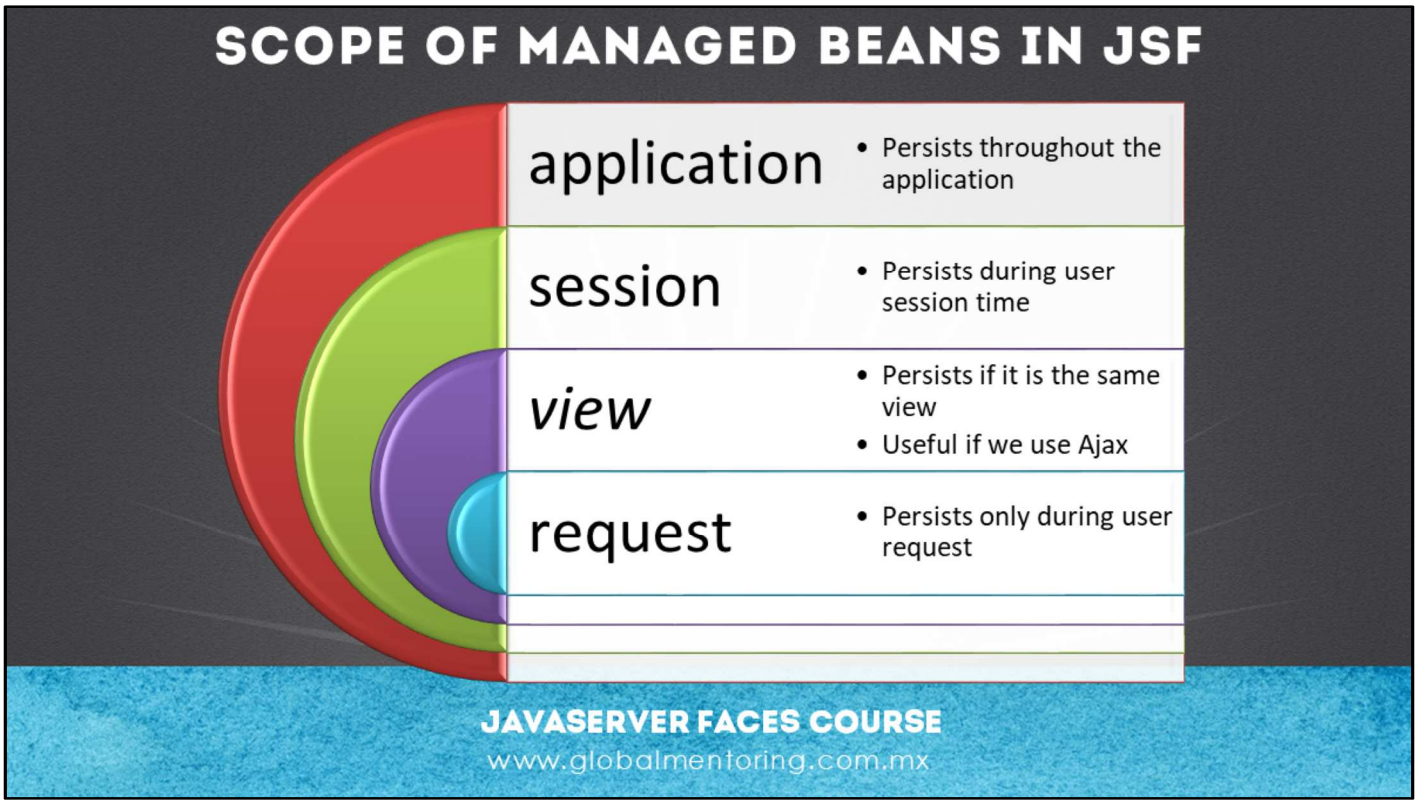

A JavaBean is a Java class that follows the following conventions:

- Empty constructor
- Private class attributes
- For each attribute, the getters and setters methods are created

The goal of Managed Beans is to control the status of web pages. JSF automatically manages the Managed Beans:

- Create the instances
  - Therefore the need for an empty constructor
- Control the life cycle of the bean
  - JSF determines the scope or scope (request, session, application, etc.) of each Managed Bean
- Call the getters or setters methods
  - For example: <h: inputText value = "#{person.name} when submitting will call the method setName()

Managed Beans can be declared in several ways, either using annotations or in the faces-config.xml file.




The JSF provide the following scopes or scopes to store information in the form of a map or table (key, value).

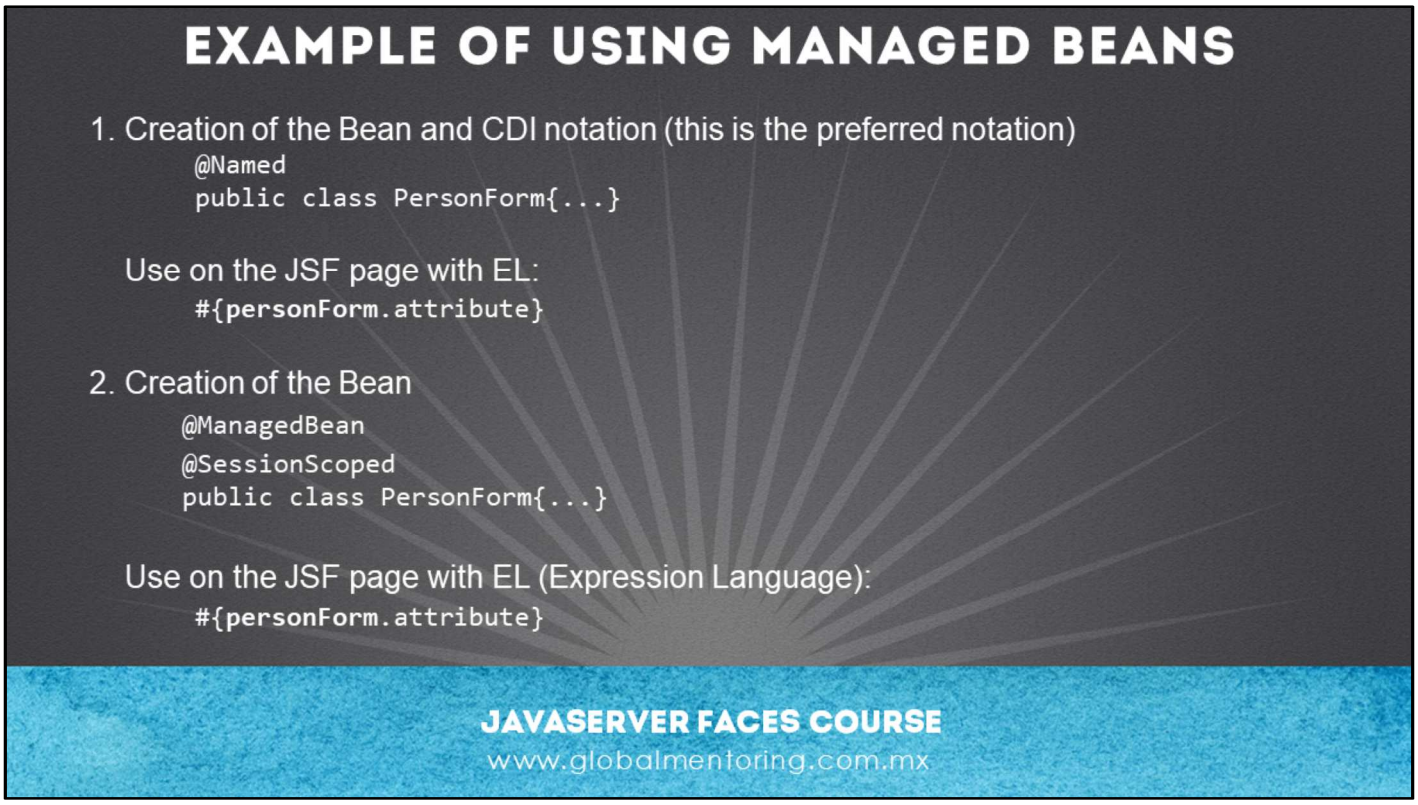
The scope of custom is simply a data structure map that links objects (values) with keys (keys). The life time of the map is managed by the implementer.

The annotations for handling JSF scopes are:

- @ RequestScoped (default scope)
- @ ViewScoped
- @ SessionScoped
- @ ApplicationScoped

Scope annotations are used after the Bean annotation, for example:

```
@Named
@SessionScoped
public class PersonForm {}
```



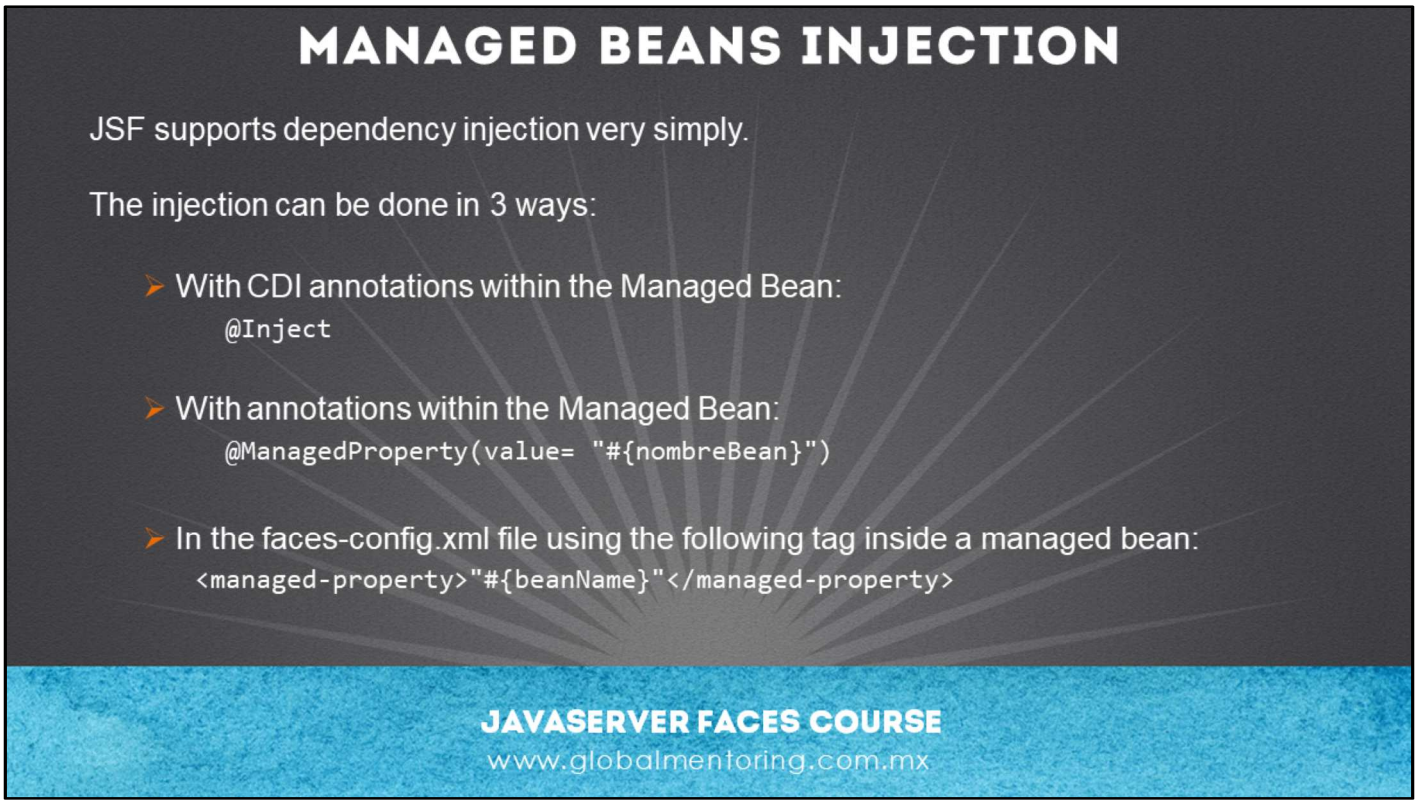

We can create Managed Beans that will later be used in JSF pages, commonly using Expression Language of JSF (EL).

In case 1, the Java EE CDI notation (Java Enterprise Version) is used. The process is very similar to the JSF notation. To configure the CDI beans, a beans.xml file must be added to the WEB-INF directory, which may be left blank. This file is used to declare specific issues in the handling of CDI.

In case 2, the JSF annotations are used. This generates an object of type PersonForm called personForm (the name starts with lowercase letters), and JSF adds this object to the session scope.

So the created bean can be used directly using EL: # {personForm.attribute}

If the deployment of our application is going to be done in a JEE Server, it is recommended to use CDI, instead if you are going to use a Web Server like Tomcat, you can use the JSF annotations.




In many occasions a JSF page can use several Managed Beans, hence the need to relate different Managed Beans.

The dependency injection (Dependency Injection - DI) is a derivative of IoC. One of the initiators of this principle was Spring framework and to this day it is a practice widely used in JEE architectures.

We can achieve dependency injection in 3 ways:

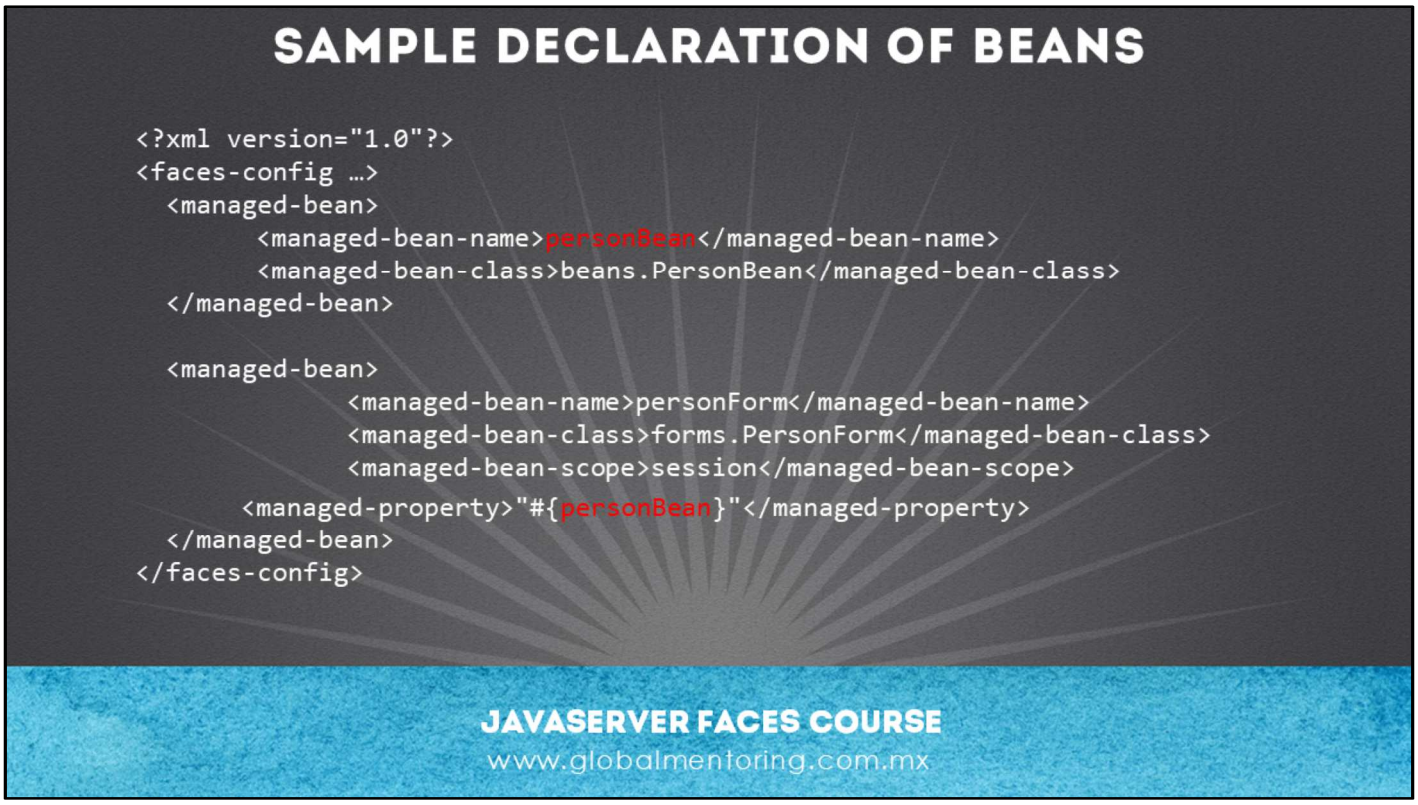
- With CDI annotations: Using the annotation @Inject on the attribute to inject the dependency.
- With annotations: Handling the annotation @ManagedProperty in the property of the Managed Bean class to inject.
- With the faces-config.xml file: Adding the <managed-property> tag inside the declared Managed Bean.


The faces-config.xml file allows us to add the JSF configuration in a centralized way, including the described features.

The general format of the JSF file depends on the version to be used, in this case it is an example of the JSF 2.3 version.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_3.xsd"
   version="2.3">
 ...
</ faces-config>
```

With the use of annotations, it is becoming less and less indispensable to use this file, although it is still used for tasks such as regional management (several languages in the web application), declaration of variables for integration with frameworks such as Spring, among other tasks.

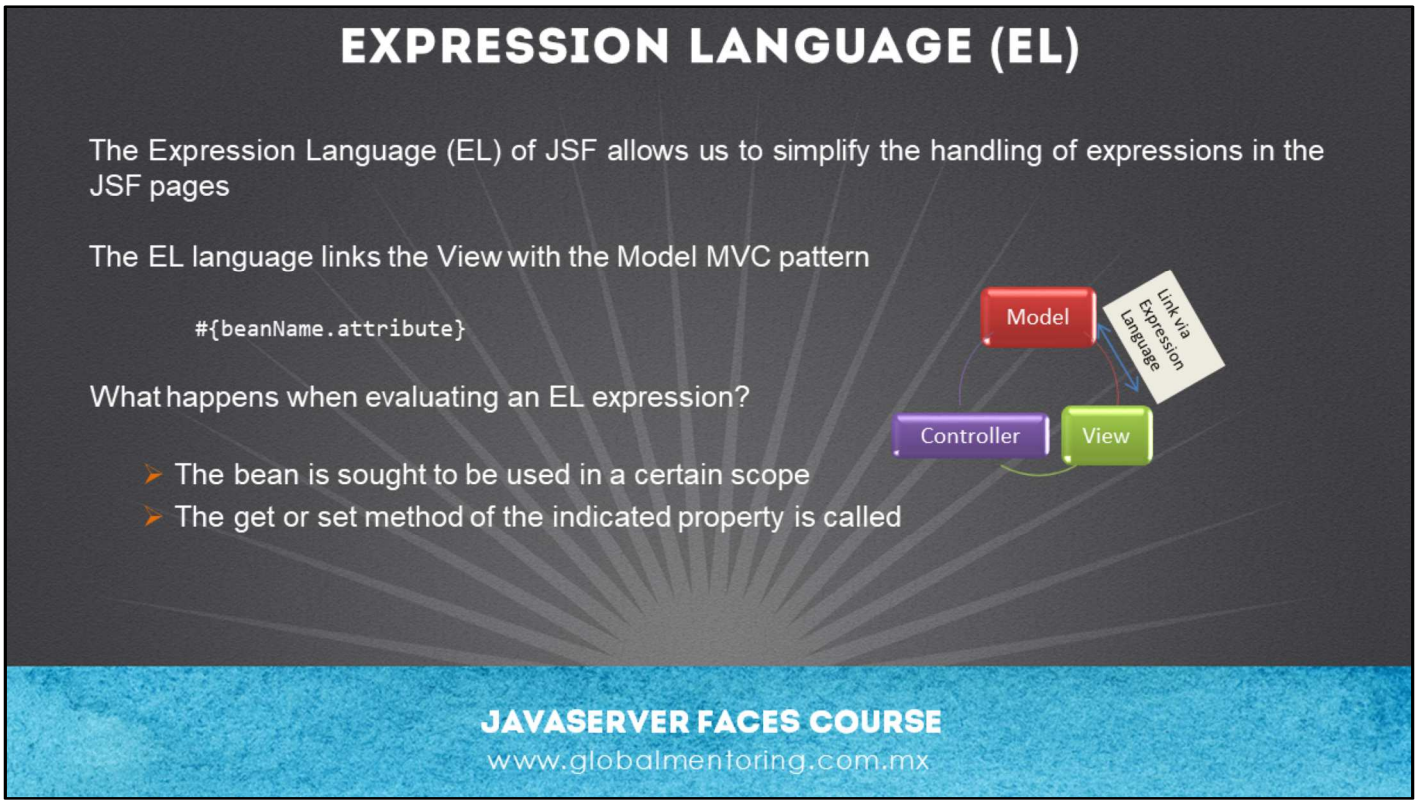



We can see an example of Managed Beans configuration using the faces-config.xml file.

The first declared bean, personBean, is in the beans package. By not defining a scope, it is created by default in the request scope.

The second declared bean (personForm) is in the forms package. This bean has a dependency with personBean, so the setPersonBean method of the PersonForm class is called to inject the dependency.

This way of configuring beans is more extensive than when using annotations, but it allows to have the configuration centralized in a single file. However, we will see that it is more common to use annotations to simplify this configuration.




The Expression (EL) language allows us to more easily evaluate and create expressions in our JSF pages.

In JSP's there is also an expression language, but it has a difference in its syntax, for example:

In JSP: $ {beanName.property}

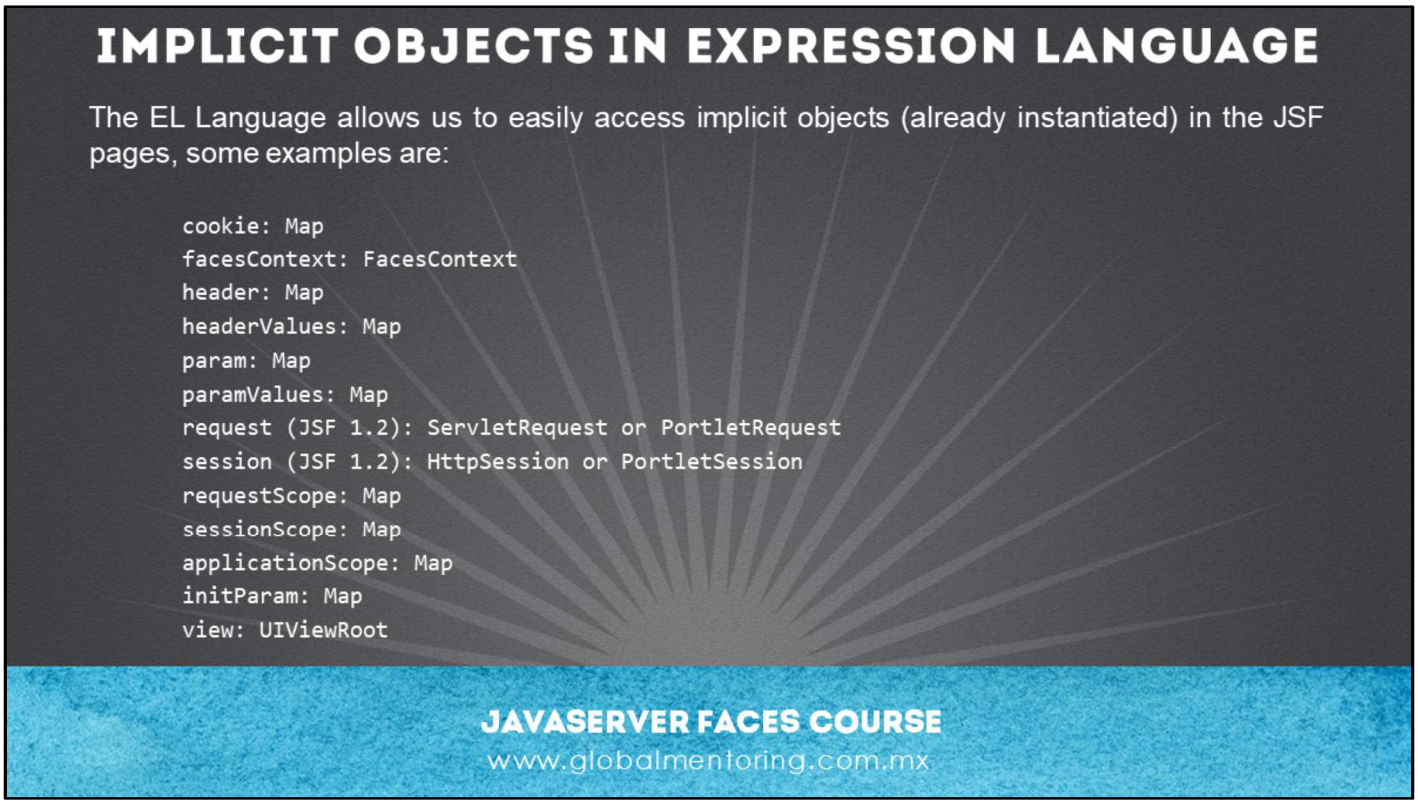
In JSF: # {beanName.property}

EL allows you to link the View with the Model, so from a JSF page we can easily access the Managed Beans of JSF.

When an EL finds a bean to use, it searches for it in a certain range. The search order is: request, view, session, application

Once the Bean to be used is located, the get or set method of the indicated property is called, for example:

```
beanName.getProperty ();
```




As in JSPs, JSF technology allows access to several of the implicit objects available on their own pages.

For example, if we want to access a bean called personBean, in the session scope, we can use any of the following notations:

1. # {personBean.property}
2. # {sessionScope.personBean.property}
3. # {sessionScope ["personBean"]. property}

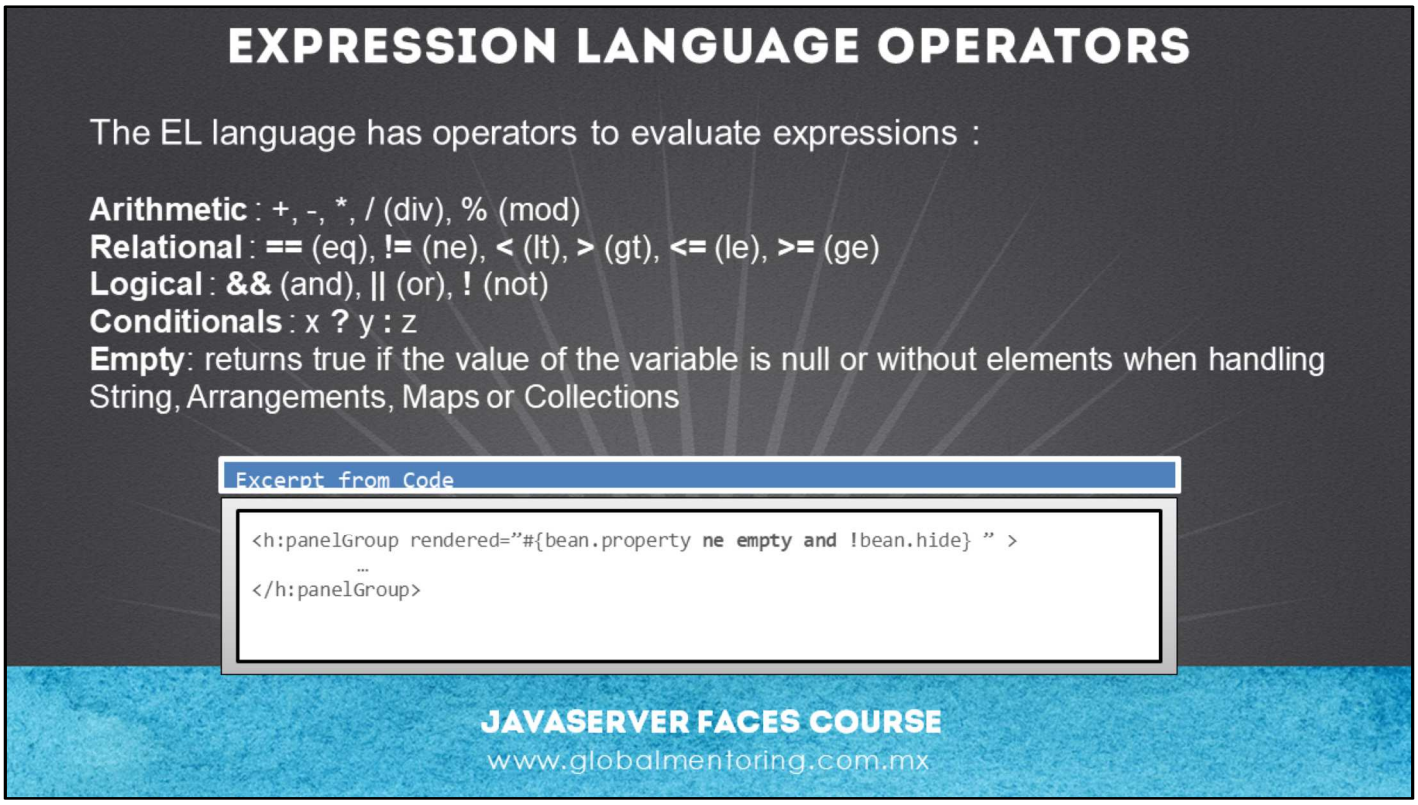
Because the employee bean is in session scope, we can retrieve it either directly or through the implicit sessionScope variable.

In addition, the sessionScope variable, being a map, we can use the arrangement notation to access its elements.

EL also allows us to access properties in a nested way, for example:

# {personBean.address.streetNumber}

In this last example, we only need to make sure that the nested values (address object) is not null, otherwise it will give a NullPointerException.




Expression Language has a series of operators to evaluate expressions, although it is considered good practice not to use this feature excessively since it breaks with the MVC design pattern when adding logic in the View.

As we can see in the code extract, the panelGroup element, which is similar to an HTML div element, is shown conditionally using the property of rendered. If the condition is true, the element is displayed.

In the example, the ne, empty, and and negation operators are used to control this logic. This is a very practical example that we can adopt to show elements conditionally on the JSF pages.