Contenido

Introducción

Este documento contiene algunos ejemplos de conceptos adicionales a la práctica que pueden ser de utilidad para el desarrollo de la práctica entregable.

Inyección de dependencias

La inyección de dependencias es una característica de utilidad para poder utilizar un Managed Bean o cualquier otro componente de JSF desde otra clase. A modo de recordatorio, los Managed Beans llevan el apellido de Managed porque se delega la gestión de la instancia en el framework. Es decir, nosotros no necesitamos implementar ningún sistema para crear, almacenar y seleccionar los objetos que vamos a utilizar en nuestra aplicación. El framework de JSF se encargará de hacer todo eso por nosotros y de ofrecernos los objetos ya creados según el scope que tengan.

Este concepto de inyección de dependencias lo utilizaremos principalmente cuando queramos acceder a propiedades y comportamientos de un Managed Bean desde otro Managed Bean. Si intentamos instanciar el Managed Bean a la hora de utilizarlo nos vamos a encontrar con el problema de que esa instancia no es la misma que se utilizará en la vista xhtml, así que no nos sirve.

Una forma de resolver esto es crear un método en el MB que quiere acceder a las propiedades de otro. Este nuevo método se tendría que llamar cuando se cree el MB y se encargará de comprobar si el objeto ya está creado. En caso de no estar creado se creará y se almacenará para que sea accesible desde otros métodos. Para que JSF sepa que tiene que invocar ese método al crear el MB, éste se tiene que anotar con la etiqueta @PostConstruct. A continuación, se muestra un ejemplo en el que un MB llamado BeanAlumnos quiere acceder a otro MB llamado BeanError. El Bean error tiene un scope de sesión, es por ello que se almacena el objeto creado en el diccionario de objetos de la sesión.

```
@Named("beanAlumno")
public class BeanAlumnos {
   ...
   private BeanError error;
   ...
   @PostContruct
   public void init() {
      error = (BeanError)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("bean
Error");
      if (error == null) {
         error = new BeanError();

FacesContext.getCurrentInstance.getExternalContext().getSessionMap().put("beanEr
ror", error);
      }
   }
   ...
}
```

Esta forma de implementar la inyección de dependencias es la que existe en JSF desde su creación. Este método tiene el problema de que acopla en exceso los dos MB. Para solucionar este problema, las nuevas versiones de JSF cuentan con otro mecanismo más sencillo para implementar la

inyección de dependencias. En la versión que utilizamos en prácticas podemos anotar el atributo que tenga la dependencia que queremos inyectar con la etiqueta @Inject. Esa anotación indica a JSF que ese atributo no será inicializado por nosotros, sino que el framework es el responsable de buscar la instancia del objeto más adecuada, crearla si no existe y ofrecérnosla a través de ese atributo. A continuación, se muestra el código de esta forma de implementación. Como se puede observar es mucho más sencillo y se evita el acoplamiento excesivo entre los dos MB al no necesitar saber detalles internos tan importantes del MB utilizado como es el caso del scope.

```
@Named("beanAlumno")
public class BeanAlumnos {
    ...
    @Inject
    private BeanError error;
    ...
}
```

Validador personalizado

En el guion de la sesión 5 se trabajó en el tema de validación de campos. El guion de esa sesión explica fundamentalmente cómo utilizar los validadores de JSF y Primefaces para la validación de los formularios. Esa es la forma preferida para hacer la validación de los campos con JSF. Sin embargo, no existen validadores para todo. Por ejemplo, si quisiésemos validar que un campo contiene sólo números pares tendríamos que desarrollar a mano nuestro propio validador.

Para implementar un validador personalizado es necesario crear una clase que implemente la interfaz Validator. esto nos obligará a implementar un método cuya signatura es validate(FacesContext context, UIComponent component, Object value), donde el primer argumento es el contexto de la aplicación que podemos utilizar para acceder a elementos conocidos como el mapa de objetos de sesión, el segundo parámetro es el componente de la vista para el que estamos realizado la validación, y, el tercero y más importante, el valor introducido por el usuario. Dentro de este método añadiremos todas las comprobaciones que deseemos. Cuando se detecte un valor que no sea válido se debe lanzar una ValidatorException que es la excepción que capturara JSF para comprobar si es válido.

Lo último que tenemos que tener en cuenta es que los validadores, al igual que los Managed Beans, los filtros y cualquier elemento de JSF es necesario registrarlos con un nombre. Al igual que el resto de los componentes, existe la forma tradicional de registrarlos en un fichero XML, pero con las versiones más modernas de JSF con las que trabajamos en clase, la forma recomendada es añadiendo la anotación @Validator sobre el nombre de la clase y asignarle entre paréntesis y comillas un nombre con el que hacerle referencia.

A continuación, hay un ejemplo de cómo se implementaría un validador de números pares en JSF.

```
@Validator("EvenValidator")
public class EvenNumberValidator {
   public validate(FacesContext context, UIComponent component, Object value) {
      String valor = (String) value;
      Pattern mask =  Pattern.compile("[0-9]*");
      Matcher matcher = mask.matcher(valor);
      if(!matcher.matches())
         throw new ValidatorException(new FacesMessage("El valor introducido
```

```
debe de ser un número"));
      int entero = Integer.parseInt(valor);
      if (entero % 2 != 0) {
         throw new ValidatorException(new FacesMessage("El valor introducido no
es par"));
      }
   }
}
```

Una vez implementado este validador se puede utilizar en la vista gracias a la etiqueta f:validator como en el siguiente ejemplo:

```
<h:inputText ... required="true">
   <f:validator validatorId="EvenValidator"/>
</h:inputText>
```

Es muy importante tener en cuenta que el valor que se coloca en el atributo validatorId debe coincidir con el nombre utilizado en la etiqueta @Validator para registrar el validador.

Posible error al mantener código comentado

JSF tiene una forma de tratar los comentarios un poco distinta a la que estamos acostumbrados con la mayoría de los lenguajes de programación. Cuando hacemos comentarios que sólo contienen un texto explicativo no hay ningún problema, pero en ocasiones, si lo que mantenemos dentro de un comentario es un fragmento de código, nos podemos encontrar con problemas, ya que, por defecto, no escapa los caracteres utilizados en las etiquetas y puede llevar a que interprete parcialmente parte del código que tengamos dentro del comentario, lo que va a producir errores al renderizar la vista.

Para evitar este comportamiento distinto hay una variable de configuración que se puede establecer en el proyecto para que ignore por completo cualquier cosa que esté dentro de los signos de comentario <!-- ... -->, incluyendo el código. Este tipo de variables de configuración se establecen en un fichero llamado web.xml que no se crea por defecto con el proyecto, así que el primer paso para resolver el problema será crear un fichero web.xml en el mismo sitio donde tenemos nuestro fichero faces_config.xml. Este fichero web.xml lo inicializaremos con el siguiente código.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
         version="6.0">

   <!-- Aquí irán las variables de configuración -->

</web-app>
```

Esta inicialización sólo incluye la definición del esquema de xml que se va a utilizar y el componente padre web-app que contendrá dentro toda la configuración. Dentro de este fichero de configuración utilizaremos el siguiente fragmento para establecer la variable de configuración FACELETS_SKIP_COMMENTS con el valor true para que se ignoren todos los comentarios de los ficheros xhtml.

```
<context-param>
   <param-name>jakarta.faces.FACELETS_SKIP_COMMENTS</param-name>
   <param-value>true</param-value>
</context-param>
```