UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualizada por:	Carlos Andrés Flórez Villarraga
Duración estimada en minutos:	90
Docente:	Christian Andrés Candela y Einer Zapata
Guía no.	19
Nombre de la guía:	Componentes de Selección y Tablas en JSF 2.3

Información de la Guía

OBJETIVO

Aprender a usar los componentes de selección (`ListBox` y `ComboBox`), tablas en JSF 2.3 y hacer uso de Convertidores `Faces`.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java EE, archivos de propiedades, XML y Glassfish.

PRECAUCIONES Y RECOMENDACIONES

Los `combobox` y los `listbox` deben ser incluidos en un formulario para su correcto funcionamiento.

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish.

EVALUACIÓN O RESULTADO

Se espera que el alumno pueda crear aplicaciones web que interactúen de forma fluida con aplicaciones de negocio por medio del uso de EJB, y haga uso de componentes especializados como las tablas y campos de selección múltiple.

Procedimiento

1. Para el desarrollo de esta guía necesitará una base de datos en MySQL, un proyecto de tipo Maven – pom, el cual contenga un proyecto Maven con soporte para el uso de JPA, uno con soporte para EJB, uno para web y otro proyecto Maven configurado para la realización de pruebas. Y una conexión a la base de datos para ser usada en la generación de las tablas.

2. Identifique en su proyecto dos entidades entre las cuales exista una relación de uno a muchos (Por ejemplo, `Producto - TipoProducto`). Teniendo en cuenta que `TipoProducto` es una entidad. Si no la tiene, créela. **NOTA:** Verifique que sus entidades tengan generados los métodos `equals`, de no ser así, deberá generar dichos métodos para sus entidades.

3. Cree o use un EJB de negocio que permita consultar todos los registros almacenados en su base de datos

de las entidades seleccionadas en el punto anterior. Si ya tiene los métodos entonces pase al siguiente punto.

4. Se debe crear una página para el registro de la entidad que se encuentra del lado de los muchos, donde se ingresen los datos básicos y seleccione el registro de la entidad padre con la cual se encuentra relacionado. Para ello se deberán ejecutar los siguientes pasos.

5. Cree un `ManagedBean` para la página creada en el punto anterior. Puede ser `ProductoBean`.

6. En el `ManagedBean` creado vamos a adicionar una instancia de nuestro EJB de negocio. Por ejemplo, para nuestro `UsuarioEJB` creamos una instancia así:

```
@EJB
private UsuarioEJB usuarioEJB;
```

Note que en este punto se está haciendo una inyección de código similar a la que se realizó en los EJB de negocio al crear el `entityManager`.

7. Adicione también a su `ManagedBean` los datos necesarios para crear un nuevo producto, los métodos `get` y `set`.

8. Cree un listado de la entidad que está del lado 1 de la relación.

```
private TipoProducto tipoProducto;
private List<TipoProducto> tiposProductos;
```

Genere los métodos getters y setters.

9. Cree un método que le permita inicializar la lista.

```
@PostConstruct
public void inicializar(){
	tiposProductos = usuarioEJB.listarTiposProductos();
}
```

NOTA: el método `listarTiposProductos` debe retornar un listado de todos los tipos de productos registrados en el sistema. Si no tiene dicho método creelo.

10. En la página de registro de producto adicione un combobox que le permita mostrar un listado de los registros de todos los tipos de productos almacenados en la base de datos así:

```
<h:selectOneMenu value="#{productoBean.tipoProducto}">
	<f:selectItems value="#{productoBean.tiposProductos}"/>
</h:selectOneMenu>
```

11. Si desea adicionar por defecto un elemento que le pida al usuario que seleccione una de las opciones lo puede hacer así:

```
<h:selectOneMenu value="#{productoBean.tipoProducto}">
      <f:selectItem itemLabel="Seleccione un tipo de producto" noSelectionOption="true"/>
      <f:selectItems value="#{productoBean.tiposProductos}"/>
</h:selectOneMenu>
```

NOTA: Como verá, basta con adicionar un elemento con un atributo que indica que no es una opción a ser seleccionada por el usuario, y se adiciona el atributo de campo obligatorio. Cree los mensajes correspondientes a las validaciones necesarias.

12. Note que se pretende asignar el tipo de producto seleccionada al atributo `tipoProducto` creado previamente en el `ManagedBean`. Sin embargo, tal como está en este momento nuestro proyecto esto no es posible, ya que el `selectOneMenu` requiere de un conversor para poder asignar el valor seleccionado. Para esto cree una clase llamada `SelectItemObjectConverter`, la cual se encargará de convertir el valor seleccionado al tipo de dato correcto.

13. Adicione a esta clase la anotación `@FacesConverter` así:

```
@FacesConverter("selectItemObjectConverter")
public class SelectItemObjectConverter implements Converter<Object>{

 @Override
 public Object getAsObject(FacesContext context, UIComponent component, String value) {

    final int index = Integer.parseInt(value);
    if (index == -1) {
        return null;
    }
    final List<?> objects = getItemsObjects(component);
    return objects.get(index);
 }

 @Override
 public String getAsString(FacesContext context, UIComponent component, Object value) {
    final List<?> objects = getItemsObjects(component);
    return String.valueOf(objects.indexOf(value));
 }

 private List<?> getItemsObjects(UIComponent component) {
    List<?> objects = Collections.emptyList();
    for (UIComponent child : component.getChildren()) {
        if (child.getClass() == UISelectItems.class) {
               objects = (List<?>)((UISelectItems)child).getValue();
        }
    }
    return objects;
 }

}
```

NOTA: Ejemplo de Convertidor tomado de https://www.adictosaltrabajo.com/. **IMPORTANTE:** Dado que el convertidor debe comparar las diferentes instancias de un objeto, en este caso `TipoProducto` debe sobreescribir el método `equals` de dicha clase para que el convertidor pueda realizar la comparación.

14. Ahora modifique la etiqueta `selectOneMenu` adicionando el siguiente atributo:

```
converter="selectItemObjectConverter"
```

15. Adicione un método a su `ManagedBean` para registrar productos, en este caso el método deberá retornar un `String` y no recibir parámetros. Por ejemplo `public String registrar()`. Por ahora el método retornará `null` sin importar si el registro fue o no exitoso.

16. Adicione un `commandButon` que invoque el método de registro creado en el punto anterior.

17. Despliegue y verifique los resultados.

18. Cree un segundo convertidor para la entidad que está del lado de los 1 en la relación. Para ello cree una clase en nuestro con el nombre `TipoProductoConverter` dicha clase debe implementar la interfaz `javax.faces.convert.Converter` así:

```
public class TipoProductoConverter implements Converter<TipoPoducto>  {

    @Override
    public Object getAsObject(FacesContext arg0, UIComponent arg1, String value) {

    }

    @Override
    public String getAsString(FacesContext arg0, UIComponent arg1, TipoPoducto value) {

    }

}
```

19. Adicionar a su convertidor las siguientes anotaciones para hacer de dicha clase un `ManagedBean`.

```
@FacesConfig(version = Version.JSF_2_3)
@Named(value = "tipoProductoConverter")
@ApplicationScoped
```

20. El método `getAsString` debe proporcionar una representación en texto del objeto, en el caso del Tipo del Producto se puede usar su código.

```
@Override
public String getAsString(FacesContext arg0, UIComponent arg1, TipoPoducto value) {
      if (value != null) {
            return value.getCodigo().toString();
      }
      return "";
}
```

21. El método `getAsObject` debe partir de la representación en `String` del objeto y devolver el objeto completo. Para ello haremos uso del `EJB` y de un método que permita buscar un `TipoProducto` por medio de su código y que retorne el objeto.

```
@Override
```

```
public Object getAsObject(FacesContext arg0, UIComponent arg1, String value) {
	TipoPoducto tipo = null;
	if (value != null && !"".equals(value)) {
		try {
			tipo = usuarioEJB.buscarTipoProducto(value);
		} catch (Exception e) {
			throw new ConverterException(new FacesMessage(arg1.getClientId() +
			":código no válido"));
		}
	}
	return tipo;
}
```

NOTA: Como se puede observar, se verifica que el `código` no sea `null` o este vacio (el `codigo` se debe convertir al tipo de dato indicado, en este caso `String`). Posteriormente se busca el `TipoProducto` con ese código. En caso de que ocurra una Excepción se crea una `ConverterException` y se genera el mensaje JSF que se quiere mostrar al usuario.

22. Modifique su menú de selección para que haga uso del nuevo convertidor. Pero ahora de la siguiente manera: `converter="#{tipoProductoConverter}"`

23. Despliegue nuevamente la aplicación y verifique su correcto funcionamiento.

24. En este punto procederemos a crear una página que permita mostrar los productos registrados en una tabla. Para ello adicione a su `ManagedBean` de Producto un método que retorne una lista de productos, el método debe retornar un `List<Producto>`.

25. Ahora cree una segunda página JSF con el nombre `listarProductos.xhtml`, en ella cree un elemento de tipo `<h:dataTable>` en su atributo `value` asigne el listado de productos de su `ManagedBean`, tal como se hizo para el menú de selección y de igual forma cree el atributo `var`. Como nuevo elemento adicione un atributo llamado `border` con valor de 1. Esto para poder observar los bordes de nuestra tabla. así:

```
<h:dataTable value="#{productoBean.productos}" var="producto" border="1">
```

26. Ahora por cada columna que quiera mostrar en la tabla adicione un elemento de tipo `h:column` así:

```
<h:column>
	<f:facet name="header">Código</f:facet>
	<h:outputText value="#{producto.codigo}"/>
</h:column>
```

El elemento `facet` permite incluir el título de la columna mientras que el `outputText` nos permite mostrar el contenido. Si queremos mostrar el nombre del tipo del Producto sería así:

```
<h:column>
	<f:facet name="header">Tipo Producto</f:facet>
	<h:outputText value="#{producto.tipoProducto.nombre}" />
</h:column>
```

27. Modifique el método que registra los productos en su `ManagedBean`. En caso de que el registro sea exitoso

debe retornar `"listarProductos"`. En caso de error retornar `null`.

28. Despliegue nuevamente la aplicación y verifique su correcto funcionamiento.

Para la próxima clase

Investigar sobre templates en JSF.