Desarrollo de LosArbolesAdm

# Desarrollo de una Funcionalidad.

1. Preparación de los paquetes:
   1. Crear en el paquete vista el paquete correspondiente a la funcionalidad a implementar. Ejemplo, si se está creando la funcionalidad de administración de arrendatarios, se debe crear el paquete: *com.spontecorp.losarboles.view.arrendatario*
   2. Crear en el paquete controller, la funcionalidad correspondiente. Ejemplo: *com.spontecorp.losarboles.controller.arrendatario*
2. En la vista se procede a crear el archivo fxml correspondiente, en este caso se llamará *ArrendatarioAdmin.fxml*. Este proceso solicita el nombre de la clase controller de esta vista, le colocamos el nombre *ArrendatarioAdminController.java* y lo colocamos en el paquete *com.spontecorp.losarboles.controller.arrendatario*, además solicitará el archivo de estilos que se usará, para ello apuntamos a nuestro archivo *LosArbolesStyles.css.*
3. Abrir en Netbeans la clase entidad correspondiente. Ejemplo: *Arrendatario.java*, estas clases se encuentra en el paquete *com.spontecorp.losarboles.model*
4. Se debe crear la clase que hará de binding de datos con el modelo y la vista, para ello en el paquete *com.spontecorp.losarboles.model.dataFx , se procede a c*rear una nueva clase Java, cuyo nombre debe ser el mismo de la entidad pero con el sufijo *Fx.* Siguiendo con el ejemplo, la clase se llamará *ArrendatarioFx.java*
5. Se crean en la nueva clase los campos *Property* que apunten a la vista, por razones de comodidad la mayoría de ellos son de tipo *SimpleStringProperty*. Como por ejemplo: *nombreProperty, apellidoProperty*, etc. Cada uno de ellos debe corresponder con su equivalente en la clase entidad. Ejemplo, si en la entidad el campo se llama *nombre* en la clase *Fx* se llamará *nombreProperty*. Se asignan todos los campos exceptuando el campo *id* de la entidad. Es recomendable que estos campos sean declarados *final*.

Esta clase debe tener además de los campos *Property*, debe tener un constructor, que asigne valor a todos esos campos. De esta manera se pueden crear los objetos *Property*, al vuelo. Cada parámetro de inicialización debe ser del mismo tipo representado en la clase entidad correspondiente.

Para el campo *status* se debe crear, en la clase *com.spontecorp.losarboles.utilities.Utilidades.java* las constantes que representarán los status del arrendatario, así como los métodos que permiten convertir de int a String y viceversa. Estos métodos se sobrecargan y llevan el siguiente nombre *parseStatusEntidad()*, donde Entidad es el status correspondiente a la entidad a la que se le están modelando los status.

Por último cada campo debe tener un método getter y setter, con la salvedad que el getter devuelve el tipo original declarado en la entidad y el setter recibe lo mismo. Ejemplo: si en la entidad el campo nombre era String, el getter retornará un String, que será el resultado de leer el valor del campo Property; y el setter recibirá como parámetro el valor String, lo que se hace es inicializar el campo Property con ese valor. No importa que los campos hayan sido declarados *final*, ya que no se está cambiando el objeto, sino su contenido.

1. En la clase controller (*ArrendatarioAdminController.java*) y una vez que la clase binding está lista, se procede a declarar los campos en el controlador que tendrán acceso a la vista, específicamente a los datos de la tabla y sus columnas. Para la TableView, se debe declarar con el genérico correspondiente a la clase binding que representará sus datos. Por ejemplo, si vamos a representar datos de tipo *Arrendatario*, se debe declarar así: *TableView<ArrendatarioFx>.*

Para las columnas, y debido a que estamos haciendo binding, declaramos el tipo Fx y su correspondiente tipo en la entidad, de la manera siguiente: *TableColumn<ArrendatarioFx, String>*

Debemos declarar un objeto de tipo ObservableList, que nos permitirá llenar la tabla. La ventaja es que ya tenemos los datos “enlazados” con su correspondiente entidad.

También declaramos el objeto facade, que nos permitirá interrelacionarnos con la base de datos, en el caso del ejemplo es *ArrendatarioFacade*.

Opcionalmente se puede crear un objeto logger, para los casos que se desee usar.

El método *initialize()* de la clase controller se usa para colocar las condiciones iniciales de la vista, en nuestro caso es llenar la tabla que mostrará los arrendatarios. Para ello debemos indicar que contendrá cada columna. Eso lo haremos *setCellValueFactory* de cada columna de la vista. A la vez creamos un método llamado *loadEntityTable()*, en el caso nuestro sustituimos el valor de Entidad por Arrendatarios, de manera que quede *loadArrendatariosTable().*

El método *loadArrendatariosTable()* será usado para recorrer una lista de los objetos traídos de la base de datos, usando la clase facade y con ellos popular la ObservableList anteriormente declarada e inicializada.

Opcionalmente es recomendable tener un método get para la ObservableList que contiene los datos, en caso que se necesite en otra parte del código.

También debemos declarar los métodos que manejarán la acción de los botones que se encuentran en la vista. Por los momentos sólo los declararemos, y luego los implementaremos, esto para probar que vamos bien. Los métodos a implementar son: *handleAgregarButton(),handleEditarButton()* y *handleInactivarButton()*.

1. Es hora de probar, para ello debemos hacer el enlace del menú de la vista principal. En la clase: *com.spontecorp.losarvoles.controller.RootLayoutController.java*, agregamos el método que nos permitirá activar esa vista cuando sea seleccionada en el menú. Ese método lo llamaremos *handleMenuArrendatarios()*. A la vez crearemos en la clase Utilidades, las constantes que nos permitirán direccionar la vista y le pasamos al método que acabamos de crear la constante y controlador correspondiente.

Hay que asegurarse que en la vista hayamos colocado que ese menú va a ser manejado por el método que recién creamos. Para ello en la vista, usando el Scene Builder, seleccionamos el objeto menú correspondiente en la vista Hierarchy y en la pestaña Code colocamos el nombre del método en la casilla On Action.

Ahora en Netbeans le damos Run al proyecto y debe aparecer nuestro sistema, y vamos al menú Arrendatarios y debemos ver una tabla vacía.

Ahora vamos a darle la funcionalidad a los botones.

1. En el paquete correspondiente, *com.spontecorp.losarboles.view.arrendatarios* en nuestro caso de ejemplo, creamos el archivo fxml que será la vista para agregar los objetos Arrendatario al sistema. Para ello creamos un nuevo archivo Empty FXML el cual llamaremos *ArrendatariosNuevoDialog*, como controlador seleccionaremos la opción que nos presenta el Netbeans, pero lo colocaremos en el paquete correspondiente, que es *com.spontecorp.losarboles.controller.arrendatario*. Como archivo css, seleccionamos nuestro archivo *LosArbolesStyles.css*, que está en la raíz del proyecto.

Primero trabajaremos con la vista. Para ello le damos doble al archivo fxml generado, de esa manera se abre automáticamente el Scene Buider y nos muestra un *AnchorPane* vacío. Se colocan todos los campos y los botones de Aceptar y Cancelar. Recordar que hay que identificar cada campo, ya que será la manera como el controller los pueda acceder.

Luego de haber hecho eso vamos al controlador de ese Dialogo. Para ello primero creamos una clase abstracta que nos servirá tanto para el dialogo de crear como el de editar. Esa clase la llamaremos, para nuestro ejemplo: *ArrendatariosDialogController.java* esta clase debe implementar la interface Inizializable. Con ella se inicializan los ChoiceBox , dos variables de control una tipo Stage, llamada *dialogStage* y una boolean llamada *okClicked*, que nos indicará si fue pulsado el botón Aceptar. Además hay que crear un objeto del tipo que se va a crear/editar, en este caso es de tipo Arrendatario.

Los métodos abstractos de esa clase serán *isInputValid()* que devuelve un boolean, para determinar si los campos fueron correctamente completados y el otro, es el método setter del objeto a crear, Arrendatario. Se hace así para que la clase concreta sea donde se use.

También se hace referencia a la acción del botón cancelar para cerrar los dialogos