**Ejemplo 1 - Parte 2 de X**

* Creación de una clase para el modelo
* Uso del modelo en una ObservableList
* Visualización del modelo mediante TableView y Controladores

**Creación de una clase para el modelo**

Necesitamos un modelo para contener la información sobre los contactos de nuestra agenda.

Añade **una nueva clase** al paquete encargado de contener los modelos (**ch.makery.address.model**) denominada **Person**.

La clase Person tendrá atributos (instancias de clase) para el nombre, la dirección y la fecha de nacimiento.

Añade el código siguiente a la clase.

**Person.java**

package ch.makery.address.model;

import java.time.LocalDate;

import javafx.beans.property.IntegerProperty;

import javafx.beans.property.ObjectProperty;

import javafx.beans.property.SimpleIntegerProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.SimpleStringProperty;

import javafx.beans.property.StringProperty;

/\*\*

\* Definir el modelo de la clase Persona.

\*/

public class Person {

private final StringProperty firstName;

private final StringProperty lastName;

private final StringProperty street;

private final IntegerProperty postalCode;

private final StringProperty city;

private final ObjectProperty<LocalDate> birthday;

/\*\*

\* Define el constructor.

\*/

public Person() {

this(null, null);

}

/\*\*

\* Constructor con algunos datos iniciales.

\*

\* @param firstName

\* @param lastName

\*/

public Person(String firstName, String lastName) {

this.firstName = new SimpleStringProperty(firstName);

this.lastName = new SimpleStringProperty(lastName);

// Damos unos valores iniciales a los datos, para realizar un chequeo.

this.street = new SimpleStringProperty("some street");

this.postalCode = new SimpleIntegerProperty(1234);

this.city = new SimpleStringProperty("some city");

this.birthday = new SimpleObjectProperty<LocalDate>(LocalDate.of(1999, 2, 21));

}

public String getFirstName() {

return firstName.get();

}

public void setFirstName(String firstName) {

this.firstName.set(firstName);

}

public StringProperty firstNameProperty() {

return firstName;

}

public String getLastName() {

return lastName.get();

}

public void setLastName(String lastName) {

this.lastName.set(lastName);

}

public StringProperty lastNameProperty() {

return lastName;

}

public String getStreet() {

return street.get();

}

public void setStreet(String street) {

this.street.set(street);

}

public StringProperty streetProperty() {

return street;

}

public int getPostalCode() {

return postalCode.get();

}

public void setPostalCode(int postalCode) {

this.postalCode.set(postalCode);

}

public IntegerProperty postalCodeProperty() {

return postalCode;

}

public String getCity() {

return city.get();

}

public void setCity(String city) {

this.city.set(city);

}

public StringProperty cityProperty() {

return city;

}

public LocalDate getBirthday() {

return birthday.get();

}

public void setBirthday(LocalDate birthday) {

this.birthday.set(birthday);

}

public ObjectProperty<LocalDate> birthdayProperty() {

return birthday;

}

}

**Explicación del código**

Con JavaFX es habitual usar Propiedades para todos los atributos de una clase usada como modelo. Una Propiedad permite, entre otras cosas, recibir notificaciones automáticamente cuando el valor de una variable cambia (por ejemplo si cambia lastName. Esto ayuda a mantener sincronizados la vista y los datos.

**LocalDate**, el tipo que usamos para especificar la fecha de nacimiento (birthday) es parte de la nueva API de JDK 8 para la fecha y la hora.

**Una lista de personas, Lista observable (ObservableList)**

Los principales datos que maneja nuestra aplicación son una colección de personas. Vamos a crear una lista de objetos de tipo **Person** dentro de la clase principal **MainApp**. El resto de controladores obtendrá luego acceso a esa lista central dentro de **MainApp**.

Las clases gráficas de JavaFX necesitan ser informadas sobre los cambios en la lista de personas. Esto es importante, pues de otro modo la vista no estará sincronizada con los datos. Para estos fines, JavaFX ha introducido nuevas clases de colección.

De esas colecciones, necesitamos la denominada **ObservableList**. Para crear una nueva ObservableList, añade el código siguiente al principio de la clase **MainApp**. También añadiremos un constructor para crear datos de ejemplo y un método de consulta (get) público:

**MainApp.java**

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import ch.makery.address.model.Person;

// ... DESPUES DE LAS OTRAS VARIABLES ...

/\*\*

\* Los datos como una observable list de Personas.

\*/

private ObservableList<Person> personData = FXCollections.observableArrayList();

/\*\*

\* Constructor

\*/

public MainApp() {

// Add some sample data

personData.add(new Person("Hans", "Muster"));

personData.add(new Person("Ruth", "Mueller"));

personData.add(new Person("Heinz", "Kurz"));

personData.add(new Person("Cornelia", "Meier"));

personData.add(new Person("Werner", "Meyer"));

personData.add(new Person("Lydia", "Kunz"));

personData.add(new Person("Anna", "Best"));

personData.add(new Person("Stefan", "Meier"));

personData.add(new Person("Martin", "Mueller"));

}

/\*\*

\* Devuelve los datos como una observable list de Personas.

\* @return

\*/

public ObservableList<Person> getPersonData() {

return personData;

}

// ... EL RESTO DE CÓDIGO DE LA CLASE ...

**Finalmente vamos a añadir datos a nuestra table.**

Para ello necesitaremos un controlador específico para la vista PersonOverview.fxml.

**Crea una clase normal dentro del paquete view** denominado **PersonOverviewController.java**. (Debemos ponerlo en el mismo paquete que **PersonOverview.fxml** o el Scene Builder no lo encontrará, al menos no en la versión actual).

Añadiremos algunos atributos para acceder a la tabla y las etiquetas de la vista. Estos atributos irán precedidos por la anotación @FXML. Esto es necesario para que la vista tenga acceso a los atributos y métodos del controlador, incluso aunque sean privados. Una vez definida la vista en fxml, la aplicación se encargará de rellenar automáticamente estos atributos al cargar el fxml. Así pues, añade el código siguiente:

**PersonOverviewController.java**

package ch.makery.address.view;

import javafx.fxml.FXML;

import javafx.scene.control.Label;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import ch.makery.address.MainApp;

import ch.makery.address.model.Person;

public class PersonOverviewController {

@FXML

private TableView<Person> personTable;

@FXML

private TableColumn<Person, String> firstNameColumn;

@FXML

private TableColumn<Person, String> lastNameColumn;

@FXML

private Label firstNameLabel;

@FXML

private Label lastNameLabel;

@FXML

private Label streetLabel;

@FXML

private Label postalCodeLabel;

@FXML

private Label cityLabel;

@FXML

private Label birthdayLabel;

// Referencia a la aplicación main.

private MainApp mainApp;

/\*\*

\* El constructor.

\* El constructor es ejecutado al ejecutar el método initialize().

\*/

public PersonOverviewController() {

}

/\*\*

\* Inicializa la clase controller. Este método se ejecuta automáticamente

\* después de haber sido cargado el fichero fxml.

\*/

@FXML

private void initialize() {

// Initialize the person table with the two columns.

firstNameColumn.setCellValueFactory(cellData -> cellData.getValue().firstNameProperty());

lastNameColumn.setCellValueFactory(cellData -> cellData.getValue().lastNameProperty());

}

/\*\*

\* Se llama a la aplicación main para devolver las referencias a si misma.

\* @param mainApp

\*/

public void setMainApp(MainApp mainApp) {

this.mainApp = mainApp;

// Añade los datos de la observable list a la tabla.

personTable.setItems(mainApp.getPersonData());

}

}

Los campos y métodos donde el archivo fxml necesita acceso deben ser anotados con @FXML. En realidad, sólo si son privados, pero es mejor tenerlos privados y marcarlos con la anotación.

El método **initialize()** es invocado automáticamente tras cargar el fxml. En ese momento, todos los atributos FXML deberían ya haber sido inicializados.

El método **setCellValueFactory(...**) que aplicamos sobre las columnas de la tabla se usa para determinar qué atributos de la **clase Person** deben ser usados para cada columna particular.

La flecha -> indica que estamos usando una característica de Java 8 denominada Lambdas. Otra opción sería utilizar un PropertyValueFactory, pero entonces no ofrecería seguridad de tipo (type-safe).

**Conexión de MainApp con PersonOverviewController**

El método **setMainApp(...)** debe ser invocado desde la clase MainApp. Esto nos da la oportunidad de acceder al objeto MainApp para obtener la lista de Person y otras cosas. Sustituye el método **showPersonOverview()** con el código siguiente, el cual contiene dos líneas adicionales:

**MainApp.java - nuevo método showPersonOverview()s**

import ch.makery.address.view.PersonOverviewController;

/\*\*

\* Muestra person overview.fxml dentro de the root layout.fxml.

\*/

public void showPersonOverview() {

try {

// Carga person overview.fxml

FXMLLoader loader = new FXMLLoader();

loader.setLocation(MainApp.class.getResource("view/PersonOverview.fxml"));

AnchorPane personOverview = (AnchorPane) loader.load();

// Carga person overview.fxml into the center of root layout.xml.

rootLayout.setCenter(personOverview);

// El controlador da acceso a la aplicación principal.

PersonOverviewController controller = loader.getController();

controller.setMainApp(this);

} catch (IOException e) {

e.printStackTrace();

}

}

**Vincular la vista al controlador**

Ahora hay que indicar a la vista declarada en **PersonOverview.fxml** cuál es su controlador y que elemento hacer corresponder a cada uno de los atributos en el controlador.

Abre PersonOverview.fxml en SceneBuilder.

Abre la sección Controller en el lado izquierdo y selecciona PersonOverviewController como controlador.

Selecciona TableView en la sección Hierarchy y en el apartado Code escribe personTable en la propiedad fx:id.

Haz lo mismo para las columnas, poniendo firstNameColumn y lastNameColumn como sus fx:id respectivamente.

Para cada etiqueta en la segunda columna, introduce el fx:id que corresponda.

Importante: En Netbeans refresca el projecto AddressApp (Botón derecho y “Clean & Build”). Esto es necesario porque a veces Netbeans no se da cuenta de los cambios realizados desde el Scene Builder.