P3. MODELOS Y VISTAS DE DATOS PARTE II

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

Índice

- Introducción
- Colecciones en JavaFX
  - ListView
  - ListView con imágenes
- Paso de parámetros a un controlador
- Ejercicio
- Aplicaciones con varias ventanas
  - Único stage y varias escenas
  - Varios stages con la correspondiente escena
- Componentes gráficos adicionales
  - TableView
  - TableView con imágenes
- Ejercicio
- Anexo. Binding de propiedades

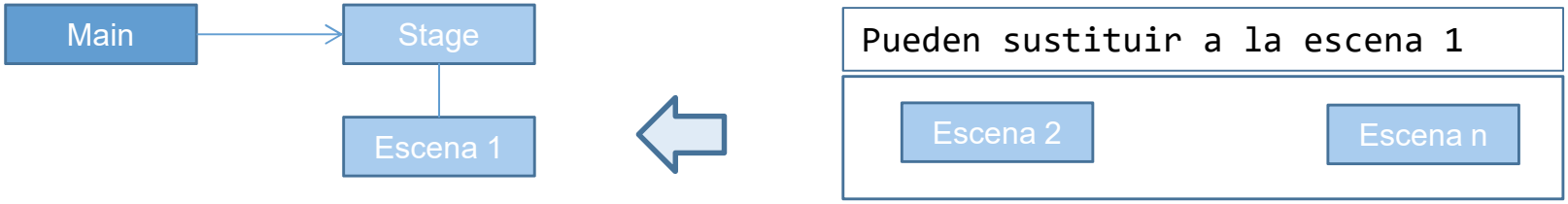
Parte I

Parte II

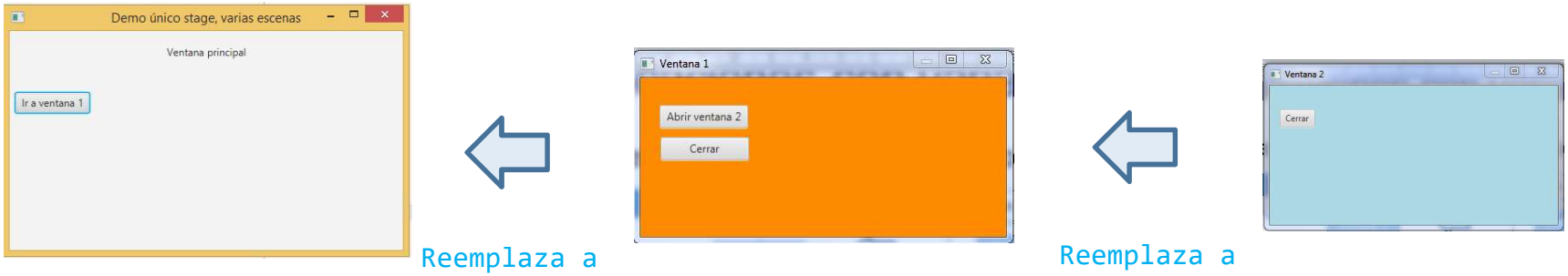
PARTE 2

Aplicaciones con varias ventanas una única visible

- Podemos tener un único Stage con varias escenas



- La aplicación tiene visible una única ventana (Stage)

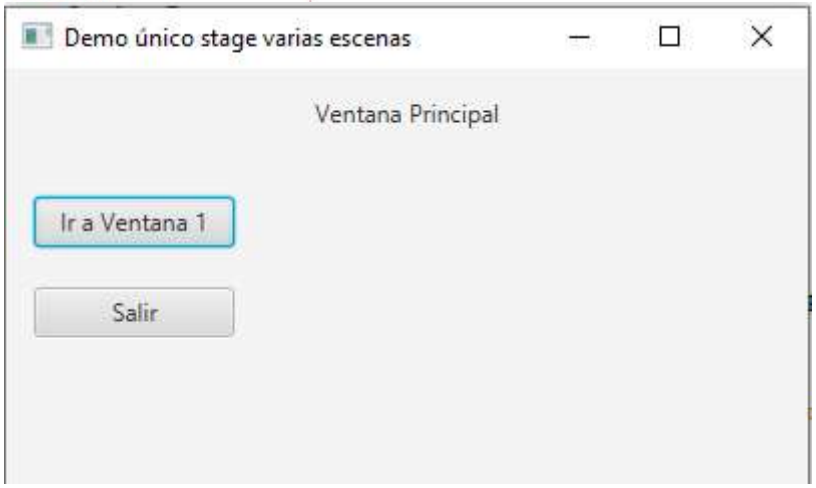


- A cada ventana se pasa el Stage principal, cada controlador carga la siguiente escena en ese stage.

Único stage varias escenas intercambiables

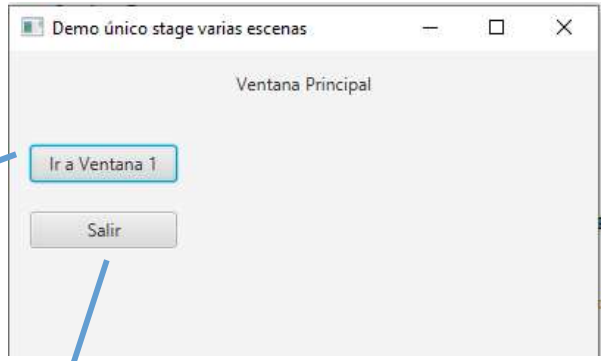
- Desde el main se carga la primera ventana y se le pasa el stage principal al controlador de la misma.

```
@Override
public void start(Stage primaryStage) {
        try {

                FXMLLoader loader = new FXMLLoader(getClass().getResource("/vista/Principal.fxml"));
                Parent root = loader.load();
                Scene scene = new Scene(root,400,400);
                primaryStage.setTitle("Demo único stage varias escenas");
                primaryStage.setScene(scene);
                // acceso al controlador
                PrincipalControlador controladorPrincipal = loader.getController();
                controladorPrincipal.initStage(primaryStage);
                primaryStage.show();
        } catch(Exception e) {
                e.printStackTrace();
        }
```



Único stage varias escenas intercambiables

- El controlador de la ventana principal de la aplicación contiene el siguiente código




```
public class PrincipalControlador implements Initializable {

private Stage primaryStage;

public void initStage( Stage stage)
{ primaryStage = stage;}

@FXML void irAVentana1(ActionEvent event) {
try { primaryStage.setTitle("Ventana 1");
  FXMLLoader miCargador  = new FXMLLoader(getClass().getResource("/vista/Ventana1.fxml"));
  Parent root = miCargador.load();
              // acceso al controlador de ventana 1
  Ventana1Controlador ventana1 = miCargador.getController()
  ventana1.initStage(primaryStage);
  Scene scene = new Scene(root,400,400);
  primaryStage.setScene(scene);
  primaryStage.show();
   } catch (IOException e) {e.printStackTrace();}
}

@FXML void pulsadoSalir(ActionEvent event) { primaryStage.hide();  }

@Override
public void initialize(URL location, ResourceBundle resources) {}

}
```

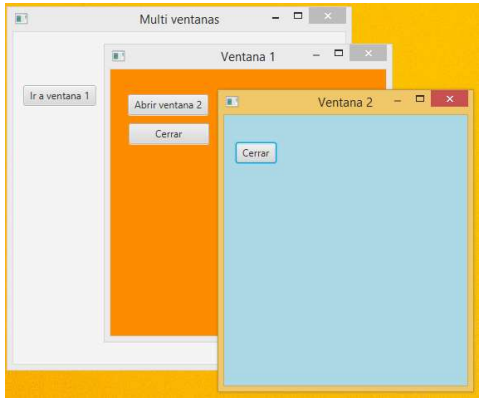
Único stage varias escenas intercambiables

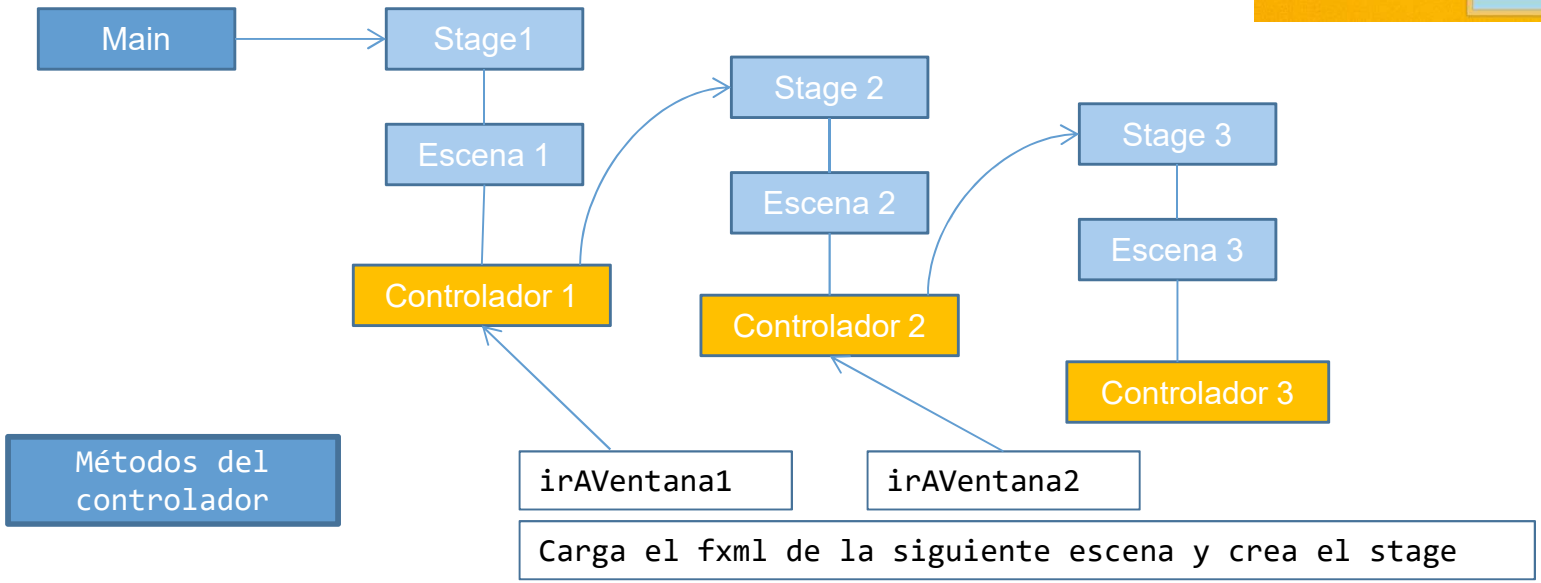
- El controlador de la segunda ventana es similar al de la primera


```
public class Ventana1Controlador implements Initializable {

    private Stage primaryStage;
    private Scene escenaPrincipal;

    public void initStage(Stage stage)
    {
        primaryStage = stage;
        escenaPrincipal = stage.getScene();
      // se obtiene la escena anterior a partir del stage
    }
    @FXML void irAVentana2(ActionEvent event) {
        // no implementado en esta versión
    }

    @FXML void cerrarAccion(ActionEvent event) {
        primaryStage.setTitle("Demo único stage varias ventanas");
        primaryStage.setScene(escenaPrincipal);
    }


    @Override
    public void initialize(URL location, ResourceBundle resources) {}

}
```

Aplicaciones con varias ventanas

- Podemos usar varios stages y cada uno con una escena
- Las tres ventanas están visibles
- Se definen modales, salvo la inicial
- Cada controlador carga el siguiente Stage





Aplicaciones con varias ventanas

- El código del main es similar al ejemplo anterior.
- Cada ventana (escena) tiene su Stage

```
public class Main extends Application {
   @Override
   public void start(Stage primaryStage) {
      try {
           FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Principal.fxml"));
           Parent root = miCargador.load();
           Scene scene = new Scene(root,400,400);
           primaryStage.setTitle("Multi ventanas");
           primaryStage.setScene(scene);
           primaryStage.show();
      } catch(Exception e) {
           e.printStackTrace();
      }
   }

   public static void main(String[] args) {
      launch(args);
   }
}
```

Aplicaciones con varias ventanas

- Controlador principal

```
public class PrincipalControlador implements Initializable {

   @FXML private void irAVentana1(ActionEvent event) {
    try {
         Stage estageActual = new Stage();

         FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Ventana1.fxml"));
         Parent root = miCargador.load();

         Scene scene = new Scene(root,400,400);
         estageActual.setScene(scene);
         estageActual.initModality(Modality.APPLICATION_MODAL);
         estageActual.show();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   @FXML void salirAccion(ActionEvent event) {
      Node n = (Node)event.getSource();
      n.getScene().getWindow().hide();
   }

}
```

Ventana 1

Modalidad

Aplicaciones con varias ventanas

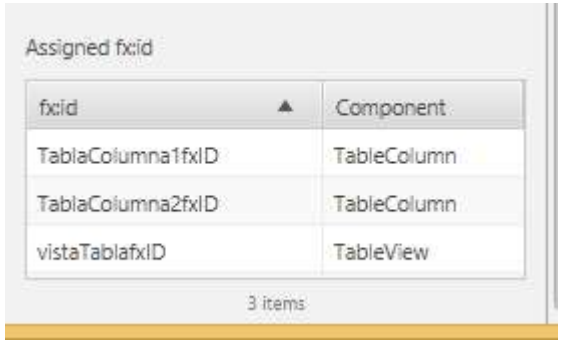
- Código controlador ventana 1

```
public class Ventana1Controlador implements Initializable {

  @FXML private void irAVentana2(ActionEvent event) {
 try { Stage estageActual = new Stage();
       FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Ventana2.fxml"));
       Parent root = miCargador.load();

       Scene scene = new Scene(root,400,400);
       estageActual.setScene(scene);
       estageActual.initModality(Modality.APPLICATION_MODAL);
       estageActual.show();
     } catch (IOException e) {e.printStackTrace();}
 }

 @FXML private void cerrarAccion(ActionEvent event) {
    Node minodo = (Node) event.getSource();
    minodo.getScene().getWindow().hide();
    System.out.println("Cerrando ventana 1");
}
}
```

Ventana 2

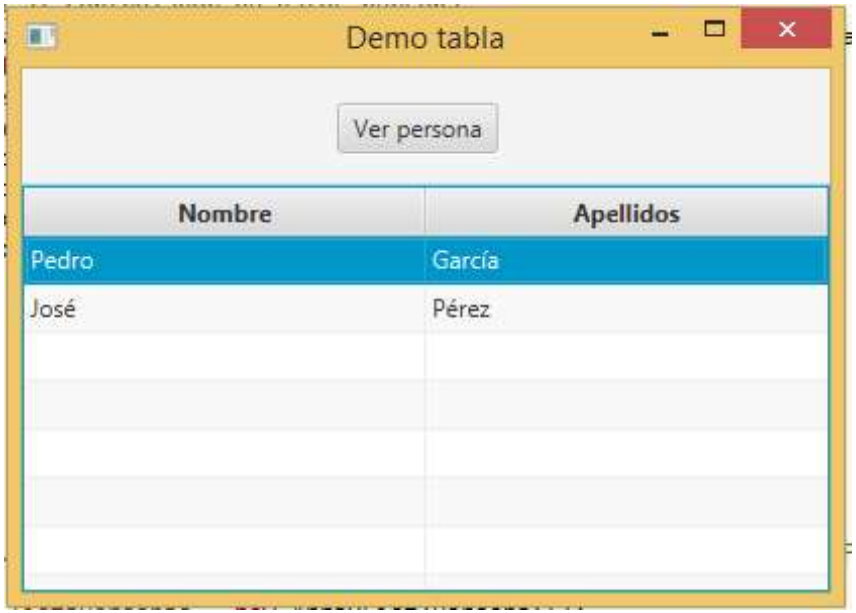
TableView

- El control está diseñado para visualizar filas de datos divididos en columnas
- TableColumn representa una columna de la tabla y contiene CellValueFactory para visualizaciones especiales, como imágenes



TableView

- La tabla contiene instancias de la clase Persona.
- Las columnas son el nombre y los apellidos




```
public class Person {

private StringProperty firstName = new SimpleStringProperty();
private StringProperty lastName = new SimpleStringProperty();

public Person(String firstName, String lastName) {
   this.firstName.setValue(firstName);
   this.lastName.setValue(lastName);
}
```

TableView

- Debemos indicar primero el tipo de los objetos que se muestran en el Tableview, y el tipo que se muestra en cada columna.
- En el controlador, generado por SceneBuilder:

```
@FXML private TableView<?> tableView;
@FXML private TableColumn<?, ?> firstNameColumn;
@FXML private TableColumn<?, ?> lastNameColumn;
```

- Se cambia a:

```
@FXML private TableView<Persona> tableView; // clase de las filas
@FXML private TableColumn<Persona, String> firstNameColumn;
@FXML private TableColumn<Persona, String> lastNameColumn;
```

Esta columna mostrará un campo de Persona

…y el campo será mostrado como un string

TableView

- Para indicar cómo se pueblan las celdas de una columna se usa el método: `setCellValueFactory` de `TableColumn`

```
private ObservableList<Persona> misPersonas;
```

- Código de inicialización en el controlador

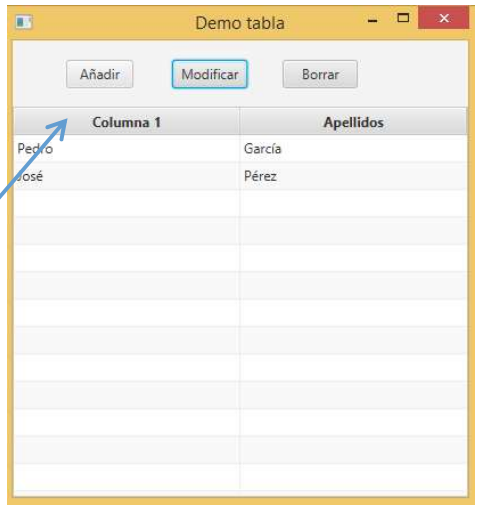
```
TablaColumna1fxID.setCellValueFactory(
                new PropertyValueFactory<Persona, String>("Nombre"));
TablaColumna2fxID.setCellValueFactory(
                 new PropertyValueFactory<Persona, String>("Apellidos"));

vistaTablafxID.setItems(misPersonas);
```

- La clase **`PropertyValueFactory<Person,String>(String prop)`**:
  - Es una clase de conveniencia para extraer una propiedad de la clase Persona
  - Internamente, tratará de invocar a <prop>Property(), get<prop> or is<prop> en el objeto Persona que debe mostrarse

TableView

- La tabla contiene instancias de la clase Persona.
- Las columnas son el nombre y los apellidos

```
public class Persona {

private StringProperty Nombre = new SimpleStringProperty();
private StringProperty Apellidos = new SimpleStringProperty();

public Persona(String nombre, String apellidos)
{
Nombre.setValue(nombre);
Apellidos.setValue(apellidos);
}
```



Código en el controlador

```
TablaColumna1fxID.textProperty().set("Columna 1");
TablaColumna1fxID.setCellValueFactory(
                new PropertyValueFactory<Persona, String>("Nombre"));
```

Indica el valor que irá en la columna

TableView

- El código:

```
firstNameColumn.setCellValueFactory(
      new PropertyValueFactory<Person, String>("firstName"));
```

- Puede originar errores de ejecución si el atributo firstName no existe. Utilice mejor una lambda expresión

```
firstNameColumn.setCellValueFactory(cellData -> cellData.getValue().NombreProperty());
```

- Equivale a:

```
firstNameColumn.setCellValueFactory(new Callback<CellDataFeatures<Person, String>,
          ObservableValue<String>>()
    {
     public ObservableValue<String> call(CellDataFeatures<Person, String> p) {
           return p.getValue().firstNameProperty();
     }
  });
```

- `CellValueFactory` indica el valor que irá en la columna, `CellFactory` indica cómo se presentará en pantalla.

TableView con imágenes

- Modificamos la tabla para que muestre una imagen y un campo (ciudad) que está en otra clase.

```
public class Person {
private final StringProperty fullName = new SimpleStringProperty();
private final IntegerProperty id = new SimpleIntegerProperty();
private final ObjectProperty<Residence> residence = new SimpleObjectProperty<>();
private final StringProperty pathImagen =  new SimpleStringProperty();
```

```
public class Residence {
private final String city;
private final String province;
```

Las clases deben tener los getters y setters para propiedades y para atributos normales

Observe que todos los atributos de Person son propiedades, más adelante se explica cómo rellenar la celdas con atributos normales

TableView con imágenes

- Campos inyectados

```
@FXML private TableColumn<Person, Integer> idColumn;
@FXML private TableColumn<Person, String> fullNameColumn;
@FXML private TableColumn<Person, Residence> cityColumn;
@FXML private TableColumn<Person, String> imageColumn;
@FXML private TableView<Person> tableView;
```

- En el controlador, en la inicialización

```
idColumn.setText("DNI");
fullNameColumn.setText("Nombre y Apellidos");
cityColumn.setText("Ciudad");
imageColumn.setText("Imagen");
// Para que se vean las columnas.
idColumn.setCellValueFactory(
                new PropertyValueFactory<Person, Integer>(“id"));
fullNameColumn.setCellValueFactory(
                new PropertyValueFactory<Person, String>("fullName"));
```

TableView con imágenes

- Para la ciudad que es un campo de Residencia, también en la inicialización del controlador

```
// ¿Qué información se visualiza?
cityColumn.setCellValueFactory(cellData3 -> cellData3.getValue().residenceProperty());

// ¿Cómo se visualiza la información?
// si quiero únicamente un string no pongo el setCellFactory
cityColumn.setCellFactory(v -> {
      return new TableCell<Person, Residence>() {
      @Override
      protected void updateItem(Residence item, boolean empty) {
           super.updateItem(item, empty);
           if (item == null || empty) setText(null);
           else setText("-->" + item.getCity());
       }
 };
});
```

Getter de las clases anteriores

Debe ser siempre un valor observable

Visualización elegida

Declarado igual que la columna correspondiente

```
@FXML private TableColumn<Person, Residence> cityColumn;
```

TableView con imágenes

- Para la columna que contiene la imagen

```
imageColumn.setCellValueFactory(celda4 -> celda4.getValue().pathImageProperty());
imageColumn.setCellFactory(columna -> {
   return new TableCell<Person,String> ()  {
      private ImageView view = new ImageView();
      @Override
      protected void updateItem(String item, boolean empty) {
        super.updateItem(item, empty);
        if (item == null || empty) setGraphic(null);
        else {
           Image image = new Image(MainWindowController.class.getResourceAsStream(item),
                                   40, 40, true, true);
           view.setImage(image);
           setGraphic(view);
        }
      }
     };
}); //setCellFactory
```

Carga el archivo png de la imagen.
item contiene el path

- El código anterior funciona si la imagen se encuentra en la carpeta `resources` del proyecto, en otro caso usar el código de la siguiente transparencia

TableView con imágenes

- Si la imagen se encuentra en una ubicación del disco duro fuera del jar del proyecto

```
imageColumn.setCellFactory(columna -> {
        return new TableCell<Person,String> ()  {
        private ImageView view = new ImageView();
        @Override
        protected void updateItem(String item, boolean empty) {
           super.updateItem(item, empty);
           if (item == null || empty) setGraphic(null);
           else {
                File imageFile = new File(item);
                //item path y nombre del archivo
                String fileLocation = imageFile.toURI().toString();
                Image image = new Image(fileLocation,40,40,true,true);
                view.setImage(image);
                setGraphic(view);
                }
           }
        };
   });
```

TableView con atributos

- Supongamos que la definición de la clase Person contiene una propiedad y 3 atributos

```
public class Person {
private StringProperty fullName = new SimpleStringProperty();
private int id;                    // atributo, no propiedad
private Residencia residence;      // no propiedad
private String pathImage;          // no propiedad
..}
```

```
public class Residence {
private final String city;
private final String province;
..}
```

- Los campos inyectados ahora son:

```
@FXML private TableColumn<Person, Integer> idColumn;
@FXML private TableColumn<Person, String> fullNameColumn;
@FXML private TableColumn<Person, String> cityColumn;
@FXML private TableColumn<Person, String> imageColumn;
@FXML private TableView<Person> tableView;
```

TableView con atributos

- Para visualizar la propiedad y los 3 atributos

```
idColumn.setCellValueFactory(cellData -> new
                SimpleIntegerProperty(cellData.getValue().getId()).asObject());

fullNameColumn.setCellValueFactory(cellData ->
                cellData.getValue().fullNameProperty());

cityColumn.setCellValueFactory( cellData -> new
           SimpleStringProperty(cellData.getValue().getResidence().getCity()));

imageColumn.setCellValueFactory(cellData -> new
           SimpleStringProperty(cellData.getValue().getPathImagen()));
```

- Para propiedades la expresión siguiente no genera ni errores de compilación, ni de ejecución, en el caso de que el nombre de la propiedad no exista. El efecto es que no muestra nada en la columna. Utilizar en su lugar la enmarcada de arriba.

```
fullNameColumn.setCellValueFactory(
                new PropertyValueFactory<Person, String>("fullName"));
```

Actividad

- A partir del proyecto de la ListView con la clase Persona, cambie la interfaz para que muestre la lista de personas en un TableView.
- Inicialice la lista de personas en el main y pase los datos al controlador.
- Añada a la interfaz los botones: Añadir, Borrar y Modificar.
  - En el caso de modificar y añadir debe mostrarse una ventana emergente para que en un caso se modifiquen los datos y en el otro se añadan.
- A realizar en el laboratorio al final de la sesión

Ejercicio continuación…

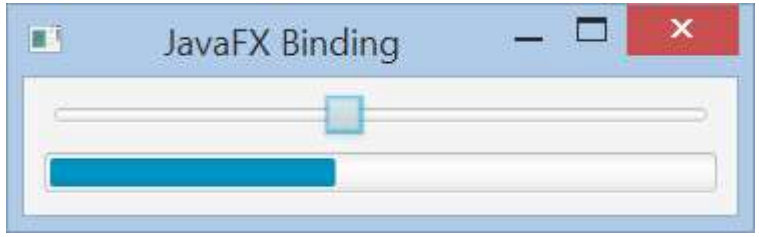
- Si terminó el ejercicio, modifíquelo para que la tabla muestre una imagen junto a cada persona.
- Las 3 imágenes están en un archivo zip de poliformat.
- En el proyecto NetBeans incluya un paquete con los 3 archivos png en un paquete recursos. Los path de las imágenes se indican:

```
"/recursos/Sonriente.png"
```

```
new Person("Juan Gómez", 45678912,
    new Residence("Valencia","Valencia"), "/recursos/Sonriente.png")
```

Anexo I: Binding de propiedades

- El binding permite sincronizar valores de propiedades, si la propiedad A está enlazada unidireccionalmente con la B, cualquier cambio de B se refleja en A. (A=f(B))
- Para crear un enlace de una única vía usaremos bind(), para crearlo de doble vía bindBidirectional(), para deshacer los enlaces unbind() y unbindBidirectional()
- Ejemplo: Enlazar la propiedad progressProperty de un ProgressBar con la propiedad valueProperty de un Slider

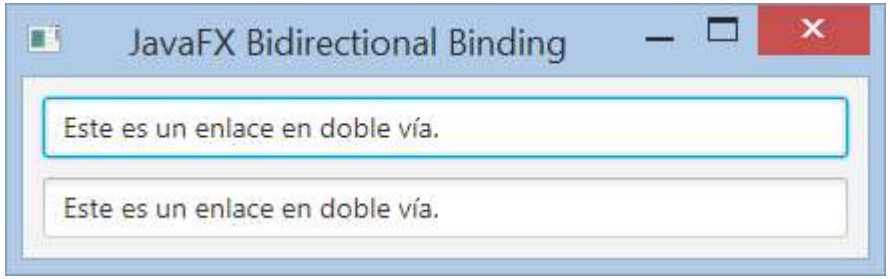


Mientras estén enlazados si se cambia por código el valor de progressProperty se produce una excepción

```
@FXML Slider slider; @FXMLProgressBar bar;

bar.progressProperty().bind(slider.valueProperty());
```

Anexo I: Binding de propiedades

- Como ejemplo enlazaremos bidireccionalmente el contenido de dos campos de texto




```
@FXML TextField tf_1;
@FXML TextField tf_2;

// en la inicialización del controlador
tf_1.textProperty().bindBidirectional(tf_2.textProperty());
```

- Cambios en uno de los campos de texto se transmiten al otro.

Anexo I: Binding numéricos de propiedades

- Se puede utilizar para enlazar valores de propiedades numéricas

```
IntegerProperty x = new SimpleIntegerProperty(100);
IntegerProperty y = new SimpleIntegerProperty(200);

NumberBinding sum = x.add(y);
int valor = z.intValue();
// sum = x+y genera un error de compilación
```

- Para acceder al valor de suma puede utilizarse: intValue(), longValue(), floatValue(), doubleValue() para obtener los valores como int, long, float y double.
- De manera equivalente

```
IntegerBinding  z = (IntegerBinding) x.add(y);
int valor = z.intValue();
```

Anexo I: Binding numéricos de propiedades

- Para el ejemplo de círculo en el gridPane, quitamos los oyentes de cambio en anchura y altura y enlazamos la propiedad radio del círuclo

```
CirculofxID.radiusProperty().bind(
      Bindings.min(gridPanefxID.widthProperty(),
                   gridPanefxID.heightProperty()).divide(5).divide(2));
```

Clase de utilidad

API Fluente, permite concatenar operaciones

```
DoubleProperty a = new SimpleDoubleProperty(1.0);
DoubleProperty b = new SimpleDoubleProperty(2.0);
DoubleProperty c = new SimpleDoubleProperty(4.0);
DoubleProperty d = new SimpleDoubleProperty(7.0);

NumberBinding result = Bindings.add (Bindings.multiply(a, b), Bindings.multiply(c,d));

NumberBinding resultado = a.multiply(b).add(c.multiply(d));
```

Referencias

ListView Oracle

https://openjfx.io/javadoc/11/javafx.controls/javafx/scene/control/ListView.html

Controles UI JavaFX Oracle

https://docs.oracle.com/javafx/2/ui_controls/overview.htm