



P3. MODELOS Y VISTAS DE DATOS

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

DOCENCIA VIRTUAL

Finalidad:

Prestación del servicio Público de educación superior (art. 1 LOU)

Responsable:

Universitat Politècnica de València.

Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:

<http://www.upv.es/contenidos/DPD/>

Propiedad intelectual:

Uso exclusivo en el entorno de aula virtual.

Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.

La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Índice

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

Parte I

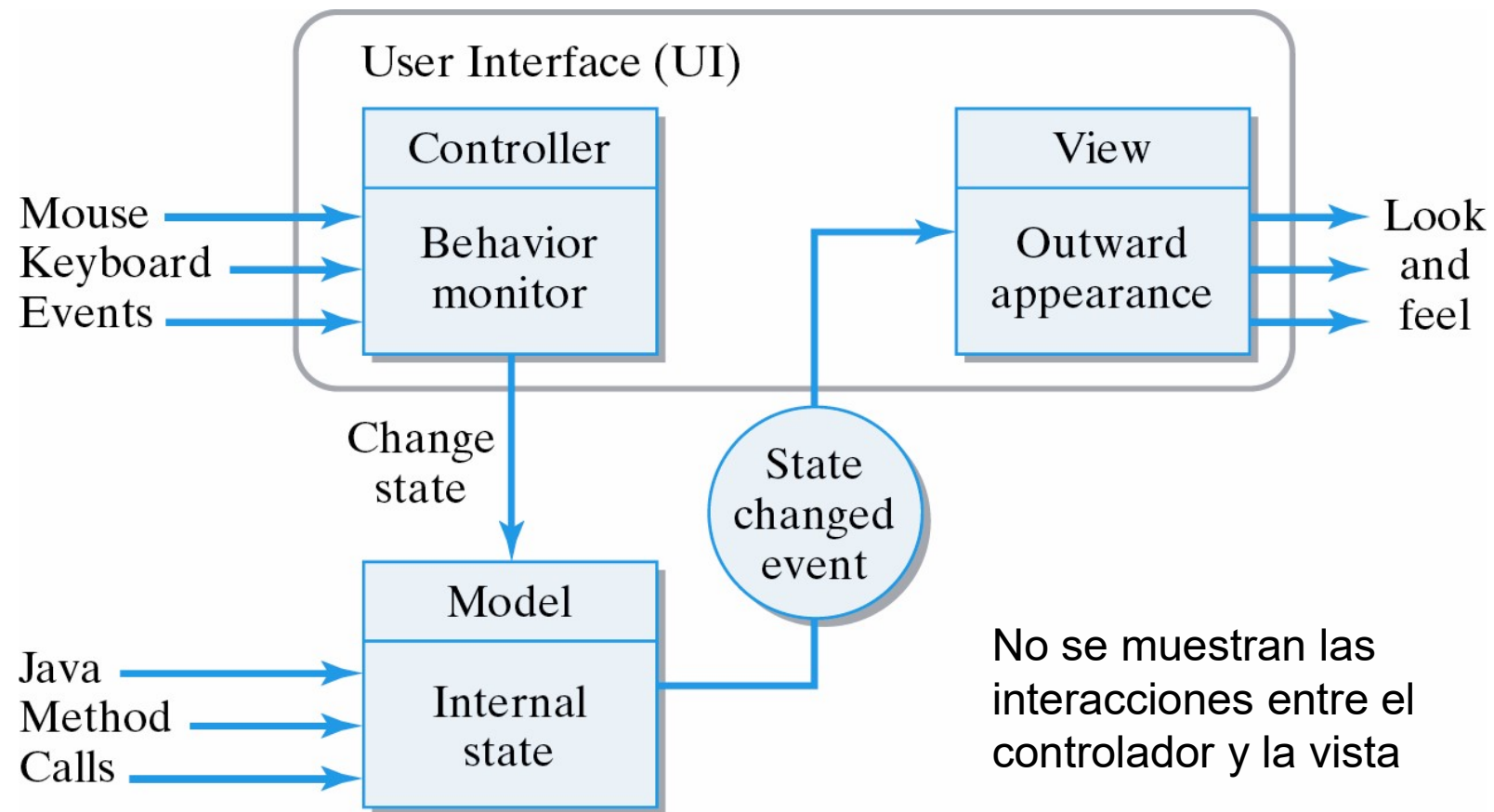
Parte II

Introducción

- Como se ha mencionado en sesiones previas las aplicaciones modernas pueden estructurarse siguiendo el patrón MVC (Modelo-Vista-Controlador)
- La arquitectura divide al sistema en 3 partes separados:
 - *Vista*: Describe cómo se muestra la información (output/display)
 - *Modelo*: ¿En qué estado está? ¿Qué datos maneja?
 - *Controlador*: ¿Qué entradas del usuario acepta y qué hace con ellas? (entrada/eventos)
- La arquitectura MVC proviene de *Smalltalk-80*, desarrollado durante los años 70.
 - en Smalltalk, MVC se utilizó como un modelo de arquitectura a nivel de aplicación: los datos (*modelo*) se hacen independientes de la UI (*vista* y *controlador*)

Introducción

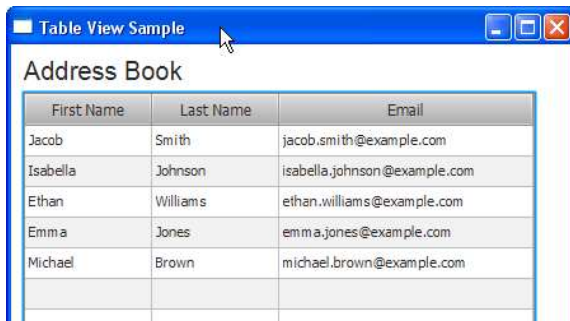
- Relaciones en la arquitectura



Introducción

- JavaFX contiene controles específicos para presentar datos en una interfaz de usuario:
 - ComboBox<T>, ListView<T>, TableView<T>, TreeTableView<T>
- Podemos separar la definición del componente (**vista**) de los datos (**modelo**) que son visualizados.
- Para el modelo se utilizan **listas observables**

Vista



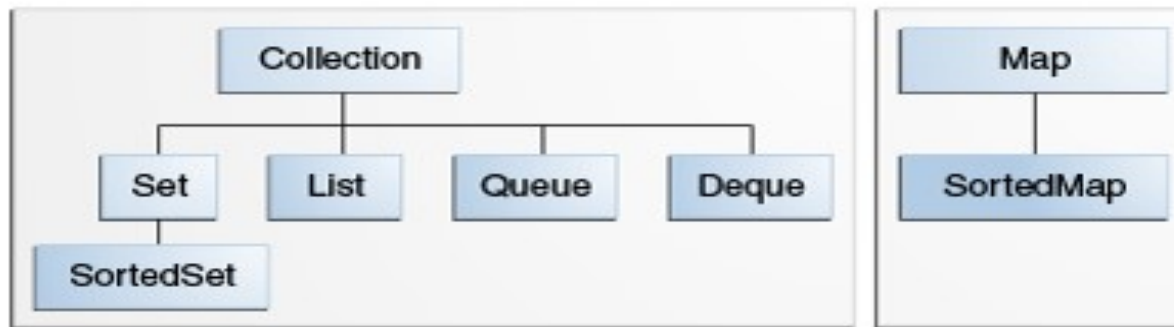
First Name	Last Name	Email
Jacob	Smith	jacob.smith@example.com
Isabella	Johnson	isabella.johnson@example.com
Ethan	Williams	ethan.williams@example.com
Emma	Jones	emma.jones@example.com
Michael	Brown	michael.brown@example.com

Modelo

```
final ObservableList<Person> data =  
FXCollections.observableArrayList(  
    new Person("Jacob", "Smith", "jacob.smith@example.com"),  
    new Person("Isabella", "Johnson", "isabella.johnson@example.com"),  
    new Person("Ethan", "Williams", "ethan.williams@example.com"),  
    new Person("Emma", "Jones", "emma.jones@example.com"),  
    new Person("Michael", "Brown", "michael.brown@example.com") );
```

Colecciones en Java

- Las colecciones de **Java** se definen a partir del siguiente conjunto de interfaces:



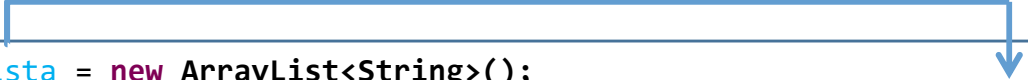
Interface	Hash	Array	Tree	Linked list	Hash+ Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Colecciones en JavaFX

- Además de las colecciones habituales de Java, JavaFX introduce nuevas: `ObservableList`, `ObservableMap`
- Interfaces
 - `ObservableList`: Una lista que permite a los oyentes monitorizar los cambios cuando éstos ocurren.
 - `ListChangeListener`: Una interface que recibe notificaciones de cambios en una `ObservableList`
 - `ObservableMap`: Un mapa que permite a los observadores monitorizar cambios cuando éstos ocurren.
 - `MapChangeListener`: Una interface que recibe notificaciones de cambios en un `ObservableMap`

Colecciones en JavaFX

- FXCollections: contiene métodos estáticos que permiten envolver colecciones de Java en colecciones JavaFX observables, o crear directamente estas últimas



```
List<String> lista = new ArrayList<String>();
ObservableList<String> listaObservable = FXCollections.observableList(lista);

listaObservable.add("item uno");
lista.add("item dos");
System.out.println("Tamaño FX Collection: " + listaObservable.size());
System.out.println("Tamaño lista: " + lista.size());
```

- La ejecución muestra
- Los elementos que se añaden a la lista son visibles desde la FXCollection

```
Tamaño FX Collection: 2
Tamaño lista: 2
```


Colecciones en JavaFX

- La colección observable es una clase **envoltorio** de la lista



- Para que los oyentes de la colección JavaFX puedan detectar cambios en la colección los elementos deben añadirse directamente sobre la `listaObservable`

Colecciones en JavaFX

- Podemos añadir un oyente a la lista observable, permitirá detectar los cambios en la misma

```
listaObservable.addListener(new ListChangeListener<String>() {  
    @Override  
    public void onChanged(ListChangeListener.Change<? extends String> arg0) {  
        System.out.println("Cambio detectado!");  
    }  
});
```

- La ejecución muestra ahora

```
listaObservable.add("item uno");  
lista.add("item dos");  
System.out.println("Tamaño FX Collection: " + listaObservable.size());  
System.out.println("Tamaño lista: " + lista.size());
```



```
<terminated> Main (1) [Java Application]  
Cambio detectado!  
Tamaño FX Collection: 2  
Tamaño lista: 2
```

Colecciones en JavaFX

- Podemos averiguar el tipo de cambio

```
listaObservable.addListener(new ListChangeListener<String>() {  
    @Override  
    public void onChanged(ListChangeListener.Change<? extends String> arg0) {  
        System.out.println("Cambio detectado!");  
        while(arg0.next())  
        {  
            System.out.println("Añadido? " + arg0.wasAdded());  
            System.out.println("Eliminado? " + arg0.wasRemoved());  
            System.out.println("Permutado? " + arg0.wasPermutated());  
            System.out.println("Reemplazado? " + arg0.wasReplaced());  
        }  
    }  
});
```

```
listaObservable.add("item uno");  
lista.add("item dos");
```

```
Cambio detectado!  
Añadido? true  
Eliminado? false  
Permutado? false  
Reemplazado? false  
Tamaño FX Collection: 2  
Tamaño lista: 2
```

Ejemplo ListView

- Las colecciones se emplean para definir el modelo de algunos componentes gráficos.
- ListView

Datos a visualizar

```
ArrayList<String> misdatos = new ArrayList<String>();  
misdatos.add("Java"); misdatos.add("JavaFX");  
misdatos.add("C++");  
misdatos.add("Python"); misdatos.add("Javascript");  
misdatos.add("C#");
```

Clase envoltorio

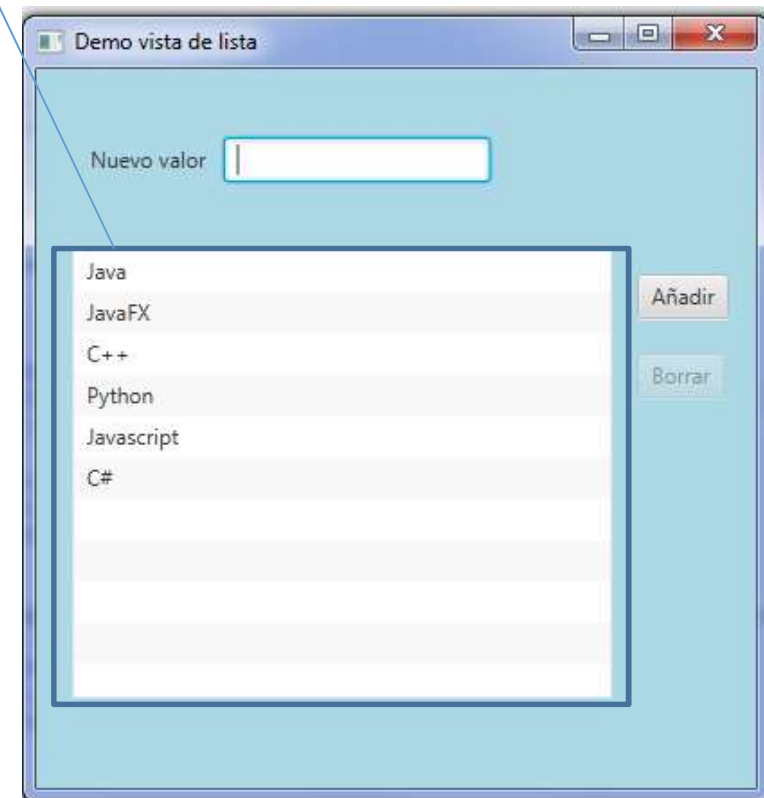
```
private ObservableList<String> datos = null;  
...  
datos = FXCollections.observableArrayList(misdatos);
```

Vinculado a la vista

```
listView.setItems(datos);
```

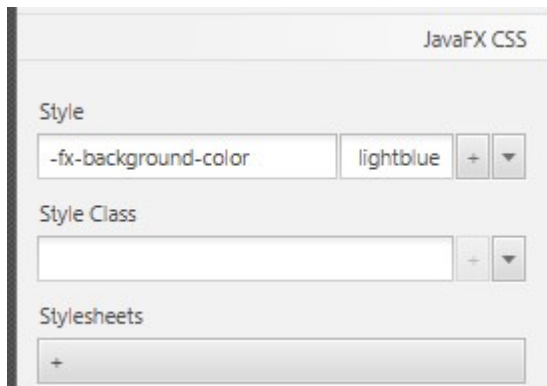
Cambios en la lista observable automáticamente provocan cambios en la vista: añadir, borrar, etc.

ListView



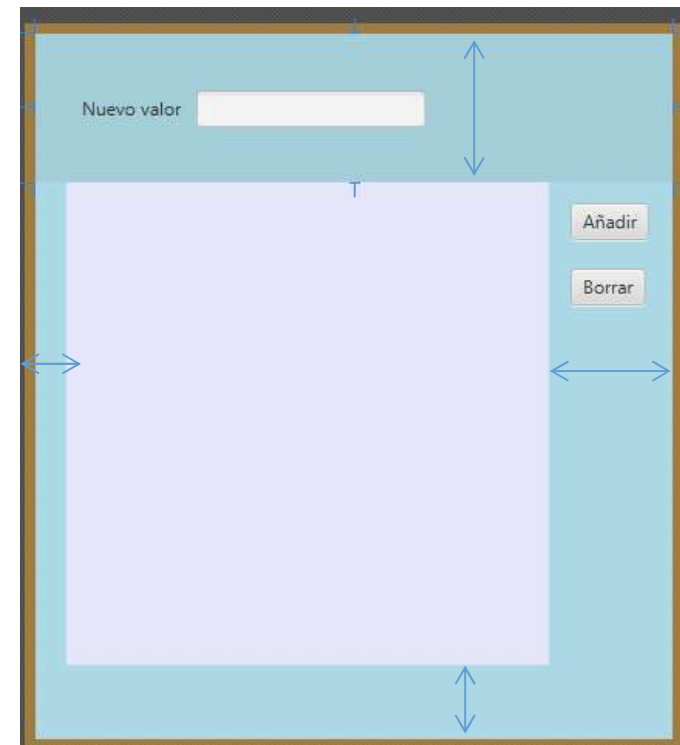
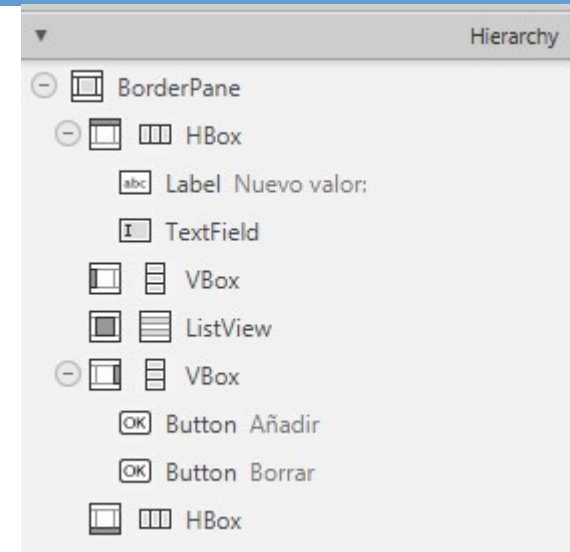
Ejemplo ListView

- Diseño de la interfaz: BorderPane, con Hbox arriba y abajo y VBox en los laterales.
- Color desde SceneBuilder con hojas de estilo CSS



- Equivalente a poner en el controlador:

```
hBoxSuperior.setStyle("-fx-background-color: lightblue;");
```



Ejemplo ListView

- Métodos útiles en ListView:
 - `getSelectionModel().getSelectedIndex()`: Devuelve el índice del elemento seleccionado de la lista, si ésta está en modo selección simple.
 - `getSelectionModel().getSelectedItem()`: Devuelve el elemento seleccionado.
 - `getFocusModel().getFocusedIndex()`: Devuelve el índice del elemento que tiene el foco.
 - `getFocusModel().getFocusedItem()`: Devuelve el elemento que tiene el foco.
- Para cambiar a modo selección múltiple:

```
getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```
- Los métodos `getSelectedIndices()` y `getSelectedItems()` de la clase `MultipleSelectionModel` devuelven listas observables que pueden usarse para monitorizar los cambios

<http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/list-view.htm#CEGGEDBF>
<http://www.java2s.com/Tutorials/Java/JavaFX>

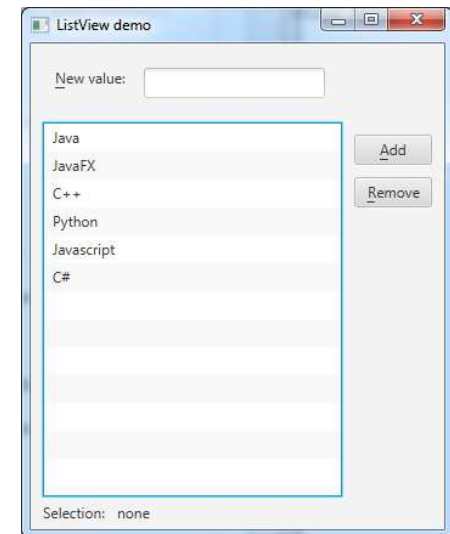
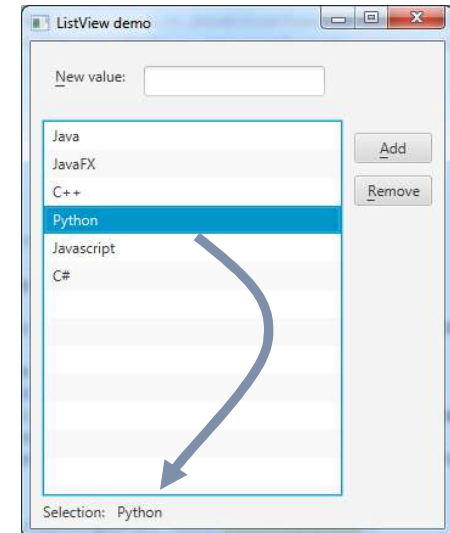
Ejemplo ListView

- Oyentes de cambios en la selección
- Opción 1

```
selectedItem.textProperty().bind(  
    listView.getSelectionModel().selectedItemProperty());
```

- Opción 2

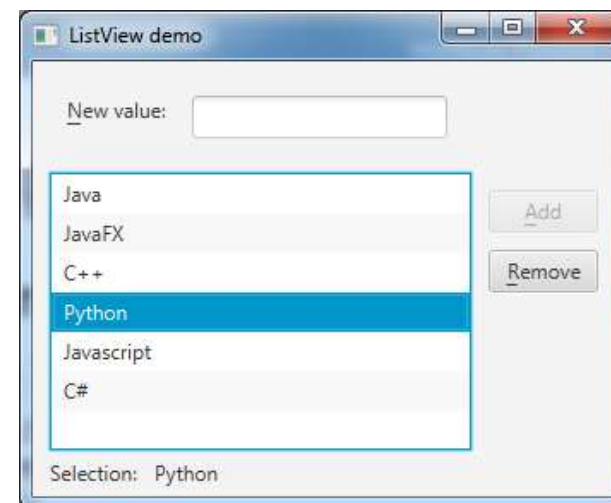
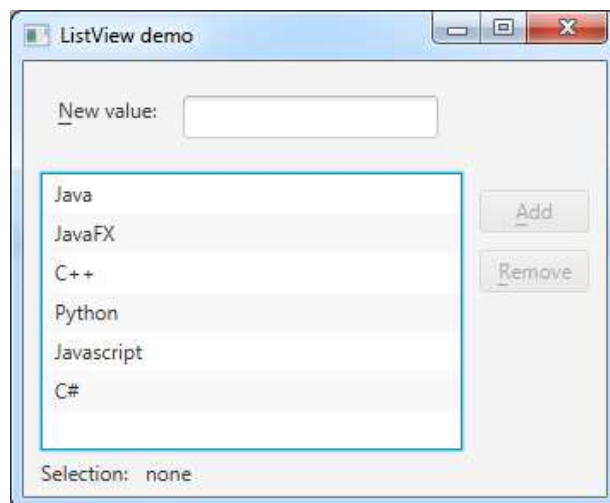
```
listView.getSelectionModel().selectedIndexProperty().  
    addListener( (o, oldVal, newVal) -> {  
        if (newVal.intValue() == -1)  
            selectedItem.setText("none");  
        else  
            selectedItem.setText(data.get(newVal.intValue()));  
    });  
selectedItem.setText("none");
```



Ejemplo ListView

- Oyentes de cambios en la selección
- Opción 3

```
selectedItem.textProperty().bind(  
    Bindings.when(listView.getSelectionModel().selectedIndexProperty().isEqualTo(-1)).  
    then("none").  
    otherwise(listView.getSelectionModel().selectedItemProperty().asString()));
```



Ejemplo ListView

- Activación/desactivación de botones al cambiar la selección

```
// The Remove button will be enabled only when an item is selected
buttonRemove.disableProperty().bind(
    Bindings.equal(-1,
        listView.getSelectionModel().selectedIndexProperty()));
```

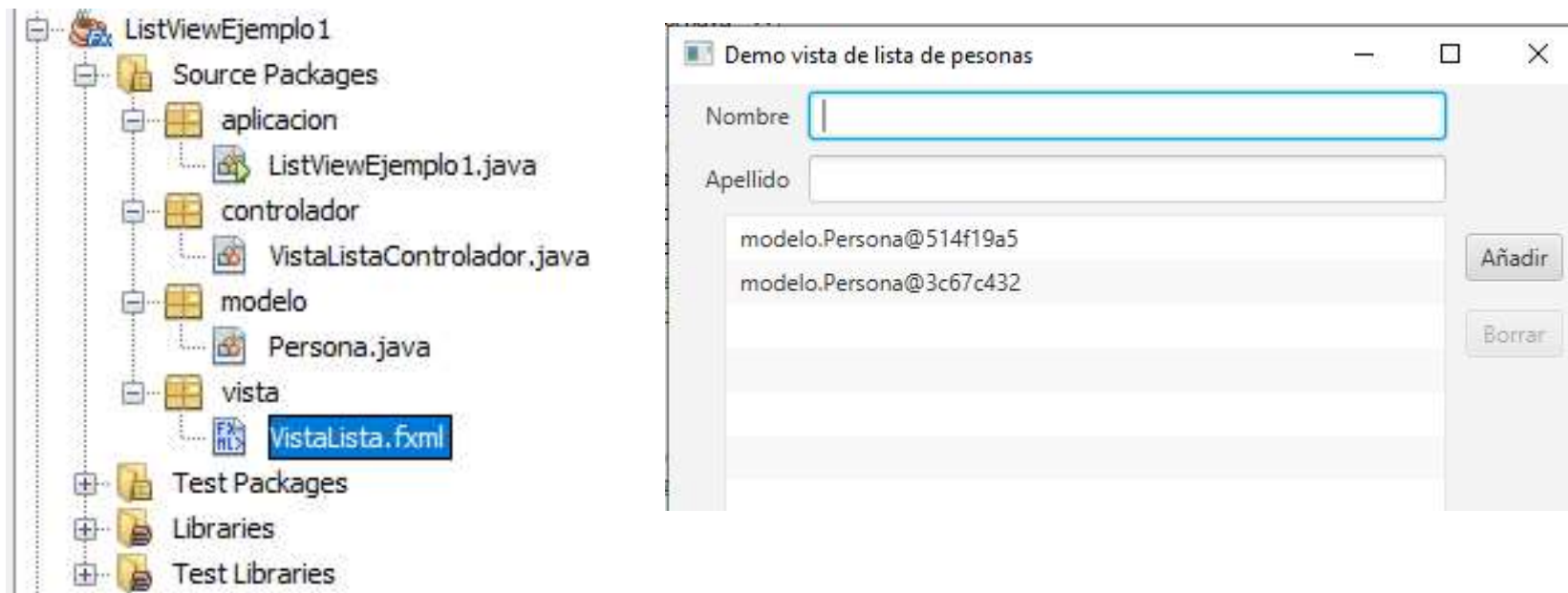
```
// The Add button will be enabled only then the TextField is not empty
buttonAdd.disableProperty().bind(firstNameText.textProperty().isEmpty())
;
```

- Los botones también puede activarse/desactivarse manualmente mediante:

```
buttonAdd.setDisable(true);
buttonRemove.setDisable(false);
```

Ejemplo ListView

- Descargue de Poliformat el ejemplo y póngalo en NetBeans, el proyecto tiene la siguiente estructura:



- Observe la descomposición en paquetes del proyecto.

Ejemplo ListView

- ListView contiene una visualización por defecto para los Strings, si se recibe un objeto se ejecuta el método `toString`.
- Cuando se necesita una visualización particular se emplean las clases `Cell` y `CellFactory`
- Todo esto es aplicable a los componentes:
 - ComboBox
 - TableView
 - TreeTableView



ListView: Cell y CellFactory

- Para la clase *Persona* tengo que indicar qué quiero que se muestre en el listView.

La clase para las celdas del listView

```
// Clase local al controlador
class PersonListCell extends ListCell<Persona>
{
    @Override
    protected void updateItem(Persona item, boolean empty)
    { super.updateItem(item, empty); // Obligatoria esta llamada
      if (item==null || empty) setText(null);
      else setText(item.getNombre() + " ," + item.getApellidos() );
    }
}
```

La información en pantalla

- En el initialize del controlador fijo la factoría de celdas

```
// en el código de inicialización del controlador
vistadeListaFXID.setCellFactory(c-> new PersonListCell());
```



ListView: Cell y CellFactory

- Si queremos añadir una imagen, en el ejemplo inicial de los lenguajes.

```
// Clase local al controlador
class LenguajeListCell extends ListCell<Lenguaje>
{
    private ImageView view = new ImageView();
    @Override
    protected void updateItem(Lenguaje item, boolean empty)
    {
        super.updateItem(item, empty);
        if (item==null || empty) {
            setText(null);
            setGraphic(null);}
        else {
            view.setImage(item.getImagen());
            setGraphic(view);
            setText(item.getNombre());
        }
    }
}
```



```
package modelo;
```

```
public class Lenguaje {
    private String nombre;
    private Image imagen;
```

```
...
```

- En el initialize del controlador:

```
listView.setCellFactory(c-> new LenguajeListCell());
```

Paso de datos a un controlador

- Supongamos que necesitamos un formulario para mostrar información de una persona, pasando como parámetro el nombre y los apellidos



TableView explicado
más adelante

```
public class DatosPersonaControlador implements Initializable {
```

```
    @FXML private TextField nombrefxID;
```

```
    @FXML private TextField apellidosfxID;
```

```
    public void initPersona( Person p)
    { nombrefxID.setText(p.getFirstName());
      apellidosfxID.setText(p.getLastName());
    }
```

```
}
```



Controlador de la interfaz Ver datos
persona

Paso de datos a un controlador

- Al cargar el fxml del formulario podemos acceder a su controlador e invocar el método que hemos llamado `initPersona`

```
//AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("/vista/DatosPersona.fxml"));
```



Cambiar por acceso no estático

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

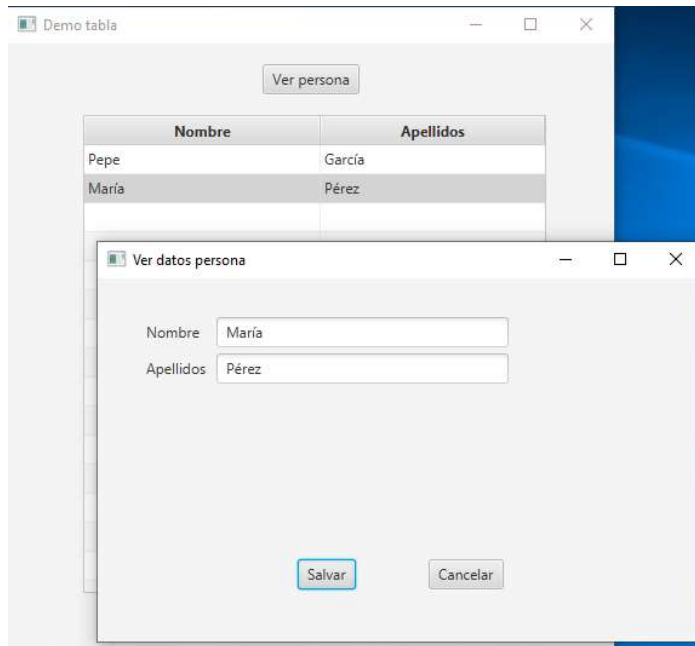
```
// acceso al controlador de datos persona  
DatosPersonaControlador controladorPersona = miCargador.<DatosPersonaControlador>getController();  
// obtiene los datos de la fila de la tabla seleccionada.  
Persona persona = VistaTablafxID.getSelectionModel().getSelectedItem();  
if (persona==null) return; // si no hay selección sale del método  
controladorPersona.initPersona(persona); // pasa los datos al segundo controlador
```

```
Scene scene = new Scene(root,400,400);  
Stage stage = new Stage();  
stage.setScene(scene);  
stage.setTitle("Ver datos persona");  
stage.show();
```

- El código anterior está en el manejador del botón Ver Persona.

Paso de datos a un controlador

- Tal como está el código de la transparencia anterior se mostrarían dos ventanas. La segunda no es modal.



Añade modalidad

```
Scene scene = new Scene(root,500,300);
Stage stage = new Stage();
stage.setScene(scene);
stage.setTitle("Ver datos persona");
stage.initModality(Modality.APPLICATION_MODAL);
//la ventana se muestra modal
stage.show();
```

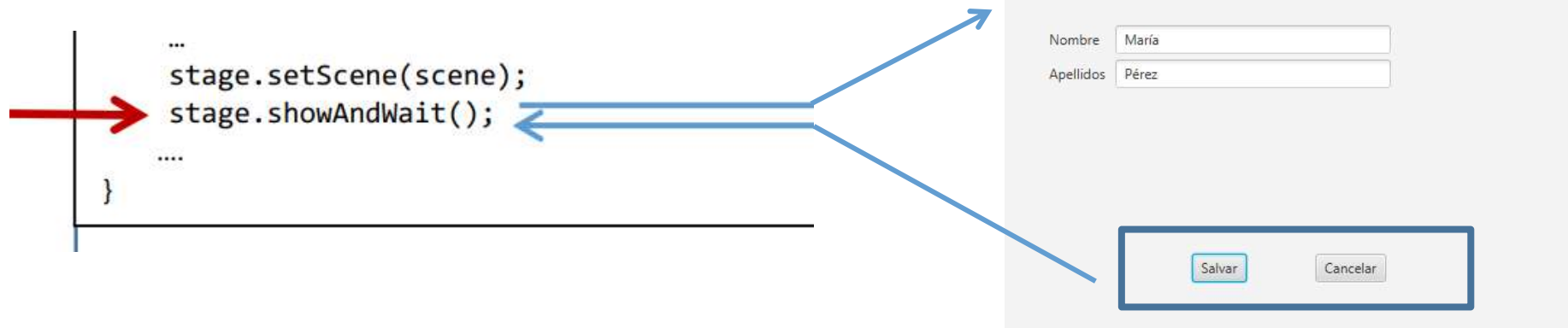

Paso de datos a un controlador

- Si queremos esperar a que la segunda ventana termine y recoger las modificaciones de la misma (nombre y/o apellidos) usaremos el método `showAndWait()`

```
@FXML
private void pulsadoVerPersona(ActionEvent event) throws IOException {
    // Abre la ventana que muestra la información de una persona.
    FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/VistaPersona.fxml"));
    Parent root = miCargador.load();
    // acceso al controlador de datos persona
    DatosPersonaControlador controladorPersona = miCargador.<DatosPersonaControlador>getController();
    // fila seleccionada de la vista de tabla.
    Persona persona = VistaTablafxID.getSelectionModel().getSelectedItem();
    // persona seleccionada en la tabla
    if (persona==null)return;
    controladorPersona.initPersona(persona);
    Scene scene = new Scene(root,500,300);
    Stage stage = new Stage();
    stage.setScene(scene);
    stage.setTitle("Ver datos persona");
    stage.initModality(Modality.APPLICATION_MODAL); //la ventana se muestra modal
    stage.showAndWait(); // espera a que se cierre la segunda ventana.
}
```

Paso de datos a un controlador

- `showAndWait()` inicia un segundo hilo de eventos anidado con el primero.



- Cuando la segunda ventana se cierra el control vuelve a la primera. Podemos añadir un método al controlador de la ventana `ver datos persona` para obtener las modificaciones hechas en la ventana emergente.

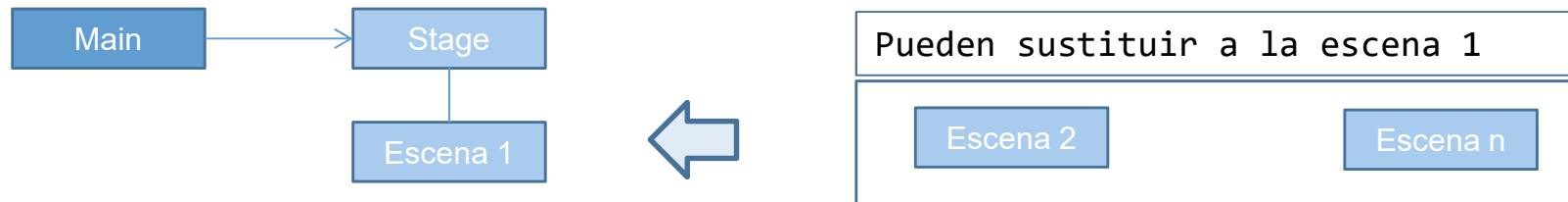
Paso de datos a un controlador

- Código para modificar

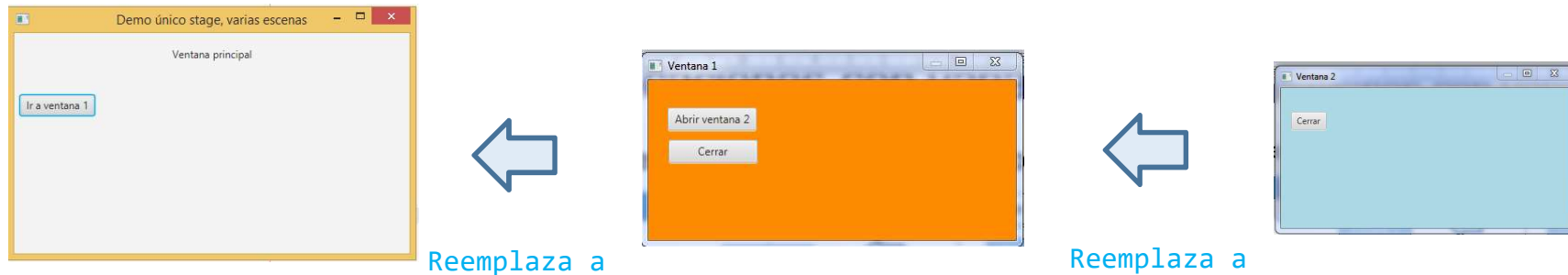
```
stage.showAndWait(); // espera a que se cierre la segunda ventana.  
// para obtener el valor modificado en la ventana emergente  
if (!controladorPersona.getCancelar())  
    {   int indice= misPersonas.indexOf(persona);  
        // índice que ocupara en la lista observable  
        Persona p= controladorPersona.getPersona();  
        //nuevo valor en el formulario emergente  
        misPersonas.set(indice, p); // actualiza la persona.  
    }
```

Aplicaciones con varias ventanas una única visible

- Podemos tener un **único Stage** con varias escenas



- La aplicación tiene visible una única ventana (Stage)

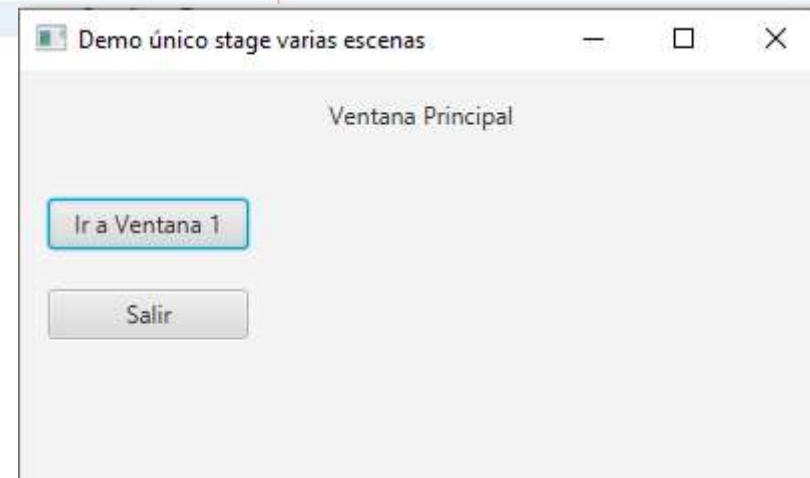


- A cada ventana se pasa el Stage principal, cada controlador carga la siguiente escena en ese stage.

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.

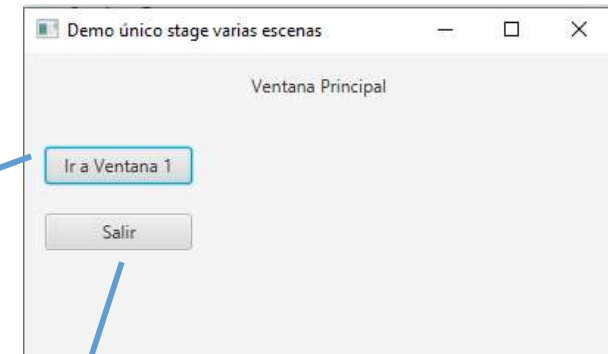
```
@Override
public void start(Stage primaryStage) {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/vista/Principal.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root, 400, 400);
        primaryStage.setTitle("Demo único stage varias escenas");
        primaryStage.setScene(scene);
        // acceso al controlador
        PrincipalControlador controladorPrincipal = loader.<PrincipalControlador>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

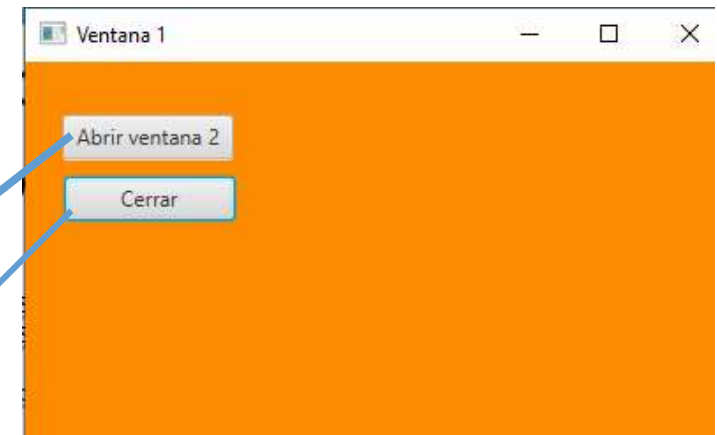
```
public class PrincipalControlador implements Initializable {  
  
    private Stage primaryStage;  
  
    public void initStage( Stage stage)  
    { primaryStage = stage;}  
  
    @FXML void irAVentana1(ActionEvent event) {  
        try { primaryStage.setTitle("Ventana 1");  
            FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Ventana1.fxml"));  
            AnchorPane root = (AnchorPane) miCargador.load();  
            // acceso al controlador de ventana 1  
            Ventana1Controlador ventana1 = miCargador.<Ventana1Controlador>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

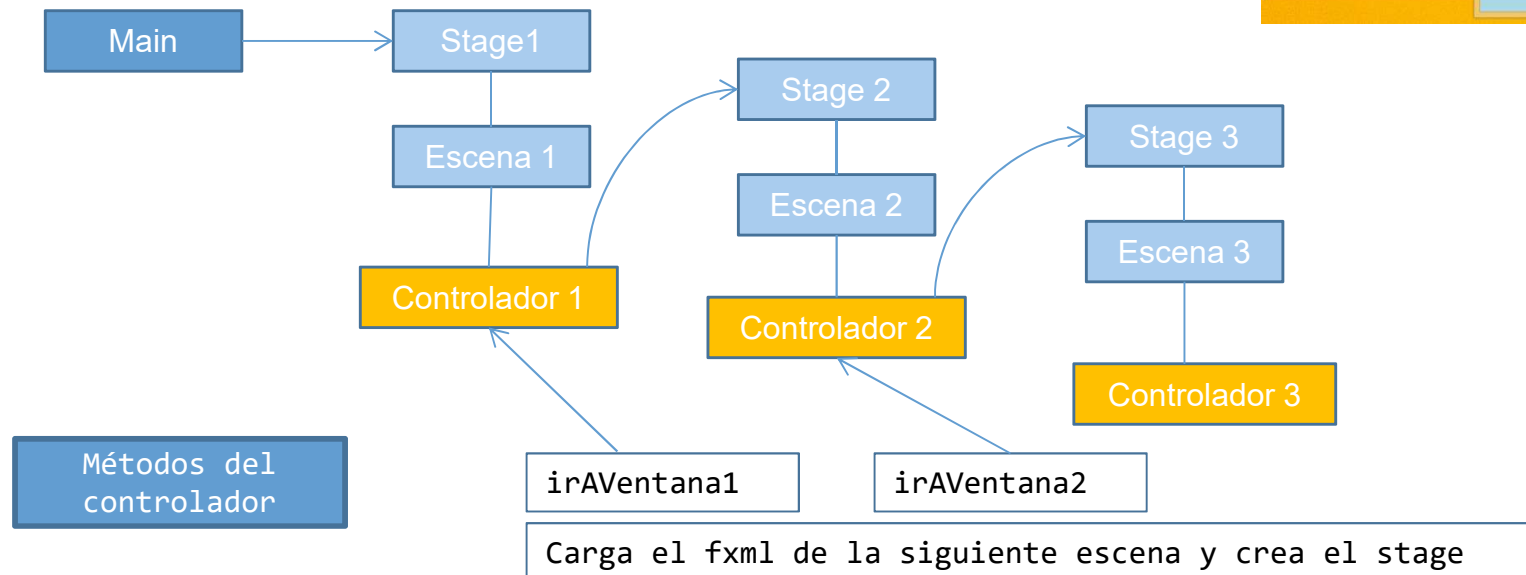
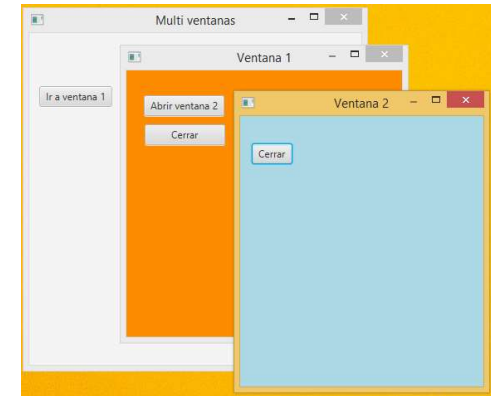
- El controlador de la segunda ventana es similar al de la primera

```
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();
```

Ventana 1

```
    FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Ventana1.fxml"));
```

```
    Parent root = miCargador.load();
```

```
    miCargador.<Ventana1Controlador>getController().initStage(estageActual);
```

```
    Scene scene = new Scene(root,400,400);
```

```
    estageActual.setScene(scene);
```

```
    estageActual.initModality(Modality.APPLICATION_MODAL);
```

Modalidad

```
    estageActual.show();
```

```
} catch (IOException e) {
```

```
    e.printStackTrace();
```

```
}
```

```
}
```

```
@FXML void salirAccion(ActionEvent event) {
```

```
    Node n = (Node)event.getSource();
```

```
    n.getScene().getWindow().hide();
```


```
}
```

```
}
```

Aplicaciones con varias ventanas

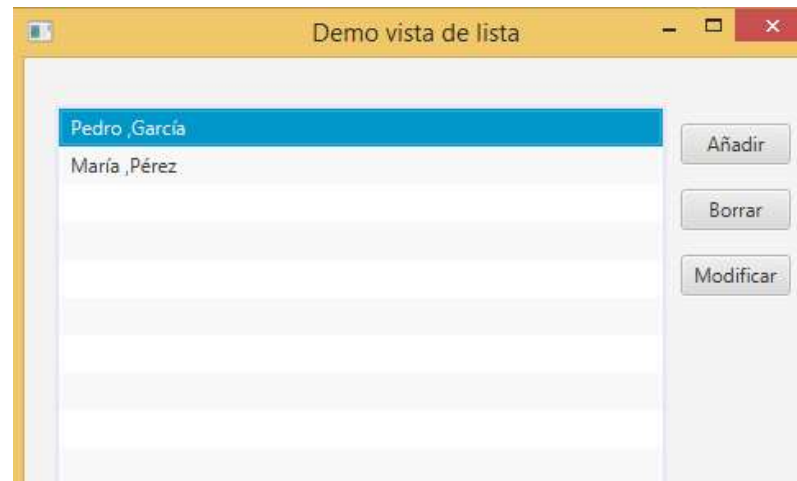
- Código controlador ventana 1

```
public class Ventana1Controlador implements Initializable {
    private Stage primaryStage;
    public void initStage(Stage stage) {
        primaryStage = stage;
        primaryStage.setTitle("Ventana 1");
    }
    @FXML private void irAVentana2(ActionEvent event) {
        try { Stage estageActual = new Stage();
            FXMLLoader miCargador = new FXMLLoader(getClass().getResource("/vista/Ventana2.fxml"));
            Parent root = miCargador.load();
            miCargador.<Ventana2Controlador>getController().initStage(estageActual);
            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");
    }
}
```



Actividad

- A partir del proyecto de la ListView con la clase Persona:
 - Crear una nueva vista con los campos Nombre y Apellido. Borrar dichos campos de la ventana original.
 - Hacer que el botón Añadir esté siempre habilitado.
 - Añadir un botón Modificar.
 - Al pulsar el botón Modificar o Añadir debe mostrarse la otra ventana para que en un caso se modifiquen los datos y en el otro se añadan.



PARTE 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

Assigned fx:id

fx:id	Component
TablaColumna1fxID	TableColumn
TablaColumna2fxID	TableColumn
vistaTablafxID	TableView

3 items

Ver persona

Nombre	Apellidos
Tabla sin contenido	

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

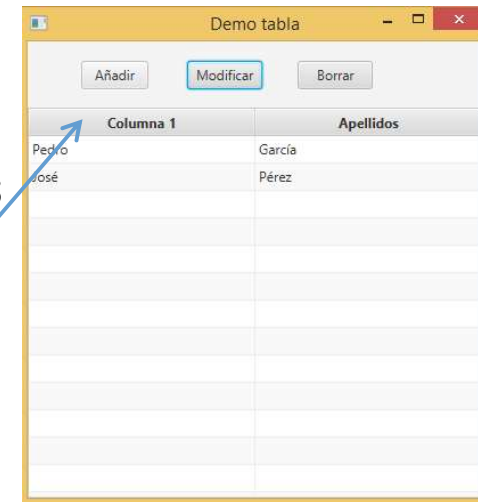
- La tabla contiene instancias de la clase Persona.
- Las columnas son el nombre y los apellidos

```
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().getNombre());
```

- 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 pathImage = new SimpleStringProperty();  
}
```

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

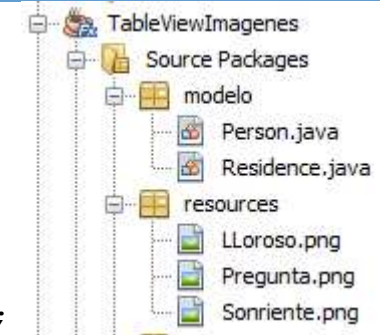
```
@FXML private TableColumn<Person, Residence> cityColumn;
```

TableView con imágenes

- Para la columna que contiene la imagen

```
imageColumn.setCellValueFactory(cellData ->
    new SimpleStringProperty(cellData.getValue().getPathImagen()));
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 {
                // el path de item es "/resources/LLoroso.png"
                Image image = new Image(item,40, 40, true, true);
                view.setImage(image);
                setGraphic(view);
            }
        }
    };
});
```

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 Residencia {  
    private String city;  
    private 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...

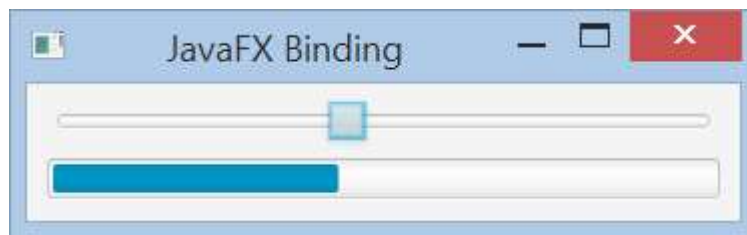
- Si terminó el ejercicio, modifíquelo para que la tabla muestre una imagen junto a cada persona.
- Las 3 imágenes están en un archivo zip de poliformat.
- En el proyecto NetBeans incluya un paquete con los 3 archivos png en un paquete recursos. Los path de las imágenes se indican:

`"/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`



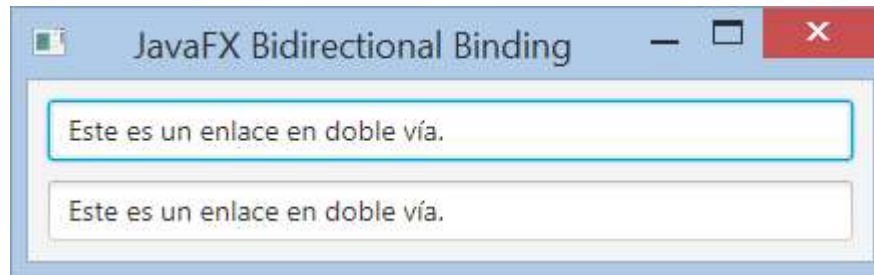
Mientras estén enlazados si se cambia por código el valor de `progressProperty` se produce una excepción

```
@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 círculo

```
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://docs.oracle.com/javafx/2/ui_controls/list-view.htm

Controles UI JavaFX Oracle

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