Advanced Programming Methods

Lecture 10 - JavaFX(continuation)

An Example based on FXML

- we use FXML to create the same login user interface,
- We separate the application design from the application logic,
- it makes the code easier to maintain.
- **`See Ex2-FXML.zip`**

An Example based on FXML

- FXMLLoader class is responsible for loading the FXML source file and returning the resulting object graph.

```
public class Main extends Application {

@Override

public void start(Stage primaryStage) {

try {

GridPane root =
(GridPane)FXMLLoader.load(getClass().getResource("Ex2.fxml"));

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

scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setTitle("FXML Welcome");

primaryStage.setScene(scene);

primaryStage.show();

} catch(Exception e) { e.printStackTrace();}

}

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

}
```

FXML file

- the GridPane layout is the root element of the FXML document and has two attributes:
  - The fx:controller attribute is required when you specify controller-based event handlers in your markup.
  - The xmlns:fx attribute is always required and specifies the fx namespace.
- The remainder of the code controls the alignment and spacing of the grid pane.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.layout.GridPane?>

<GridPane alignment="CENTER" gridLinesVisible="true" hgap="10.0"

prefHeight="292.0" prefWidth="364.0" vgap="10.0"

xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.141"

fx:controller="application.Ex2Controller">

   <padding>

      <Insets bottom="10.0" left="25.0" right="25.0" top="25.0" />

   </padding>

</GridPane>
```

Text, Label, TextField, and Password Field Controls

```
 <children>
      <Text strokeWidth="0.0" text="Welcome"
wrappingWidth="63.576171875" />
      <Label text="User Name" GridPane.rowIndex="1" />
      <Label text="Password" GridPane.rowIndex="2" />
      <TextField GridPane.columnIndex="1" GridPane.rowIndex="1" />
      <PasswordField fx:id="passwordField" GridPane.columnIndex="1"
GridPane.rowIndex="2" />
   </children>
```

Add a Button and Text

```
  <HBox alignment="BOTTOM_RIGHT" prefHeight="100.0" prefWidth="200.0" spacing="10.0" 
GridPane.columnIndex="1" GridPane.rowIndex="4">

         <children>

            <Button mnemonicParsing="false" onAction="#handleSubmitButtonAction" 
prefHeight="28.0" prefWidth="81.0" text="Sign In" />

         </children>

      </HBox>

      <Text fx:id="actionTarget" strokeType="OUTSIDE" strokeWidth="0.0" 
GridPane.columnSpan="2" GridPane.rowIndex="6" />
```

Add Code to Handle an Event

- **The controller has to be created (either a skeleton from SceneBuilder or from fxml file using the option source/generate controller) to handle the events**

```
package application;

import javafx.fxml.FXML;

import javafx.scene.text.Text;

import javafx.event.ActionEvent;

import javafx.scene.control.PasswordField;

public class Ex2Controller {

@FXML

private PasswordField passwordField;

@FXML

private Text actionTarget;

// Event Listener on Button.onAction

@FXML

public void handleSubmitButtonAction(ActionEvent event) {

actionTarget.setText("Sign in button pressed");
```

Style the Application with CSS

- Inside fxml file we can set the previous css file to obtain the same style

```
<GridPane alignment="CENTER" hgap="10.0" prefHeight="217.0"
prefWidth="300.0" styleClass="root" stylesheets="@application.css"
vgap="10.0" xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/8.0.141"
fx:controller="application.Ex2Controller">
```

FXML

FXML = is a scriptable, XML-based markup language for constructing Java object graphs

- **declarative approach** (versus programmatic)
- Tree of components
- Role separation
- Language independent (Java, Scala, Clojure, etc.)
- Support for internationalization

XML

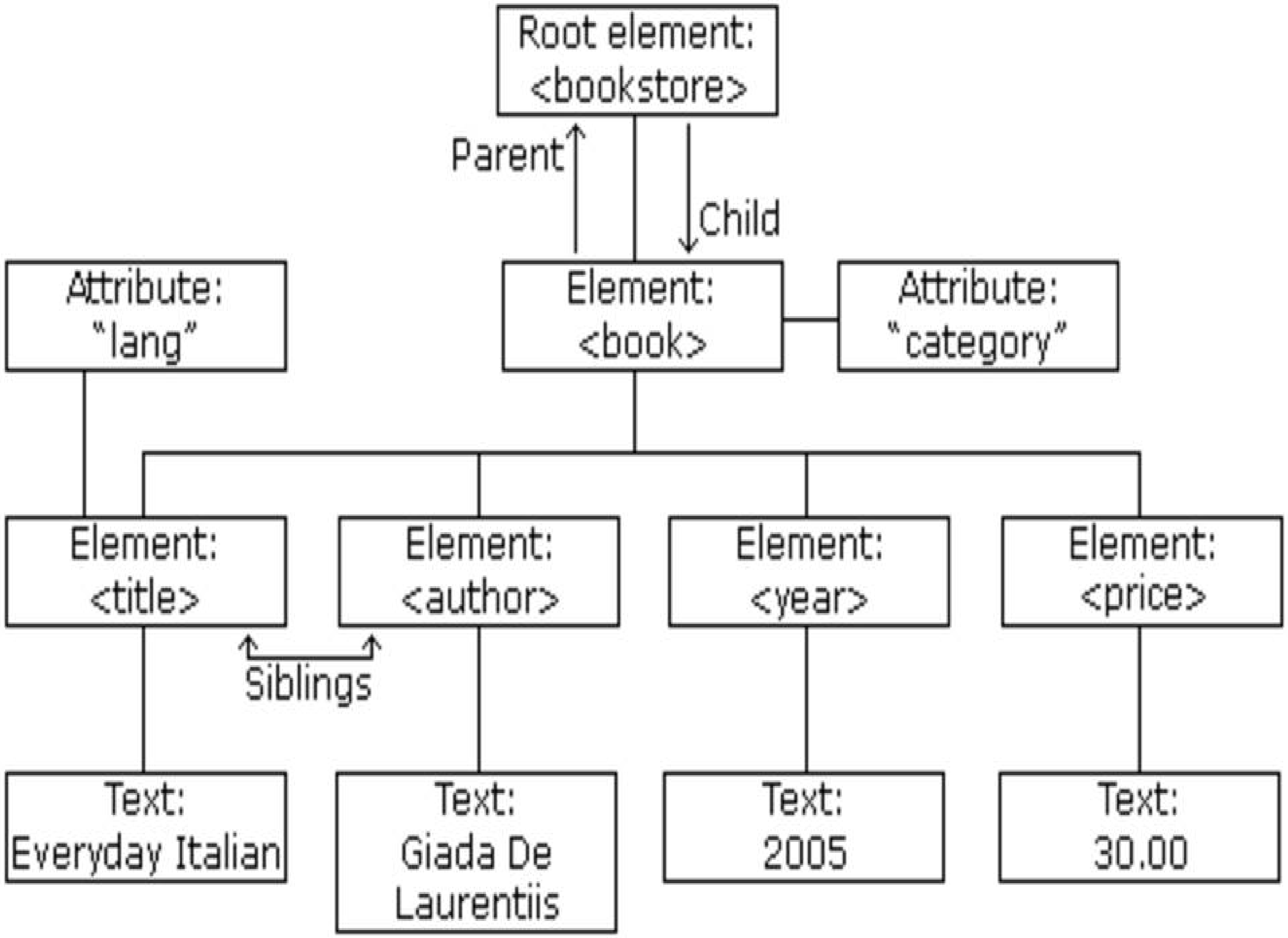
XML stands for EXtensible Markup Language.

XML was designed to store and transport data.

XML was designed to be self-descriptive- human- and machine-readable.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget!</body>
</note>
```

XML Tree Structure



Example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

FXML

From a Model View Controller (MVC) perspective:

- the FXML file that contains the description of the user interface is the view.
- The controller is a Java class (optionally implementing the Initializable class), which is declared as the controller for the FXML file.
- The model consists of domain objects, defined on the Java side, that you connect to the view through the controller.

FXML

- does not have a schema, but it does have a basic predefined structure.
- maps directly to Java
- most JavaFX classes can be used as elements
- most properties can be used as attributes

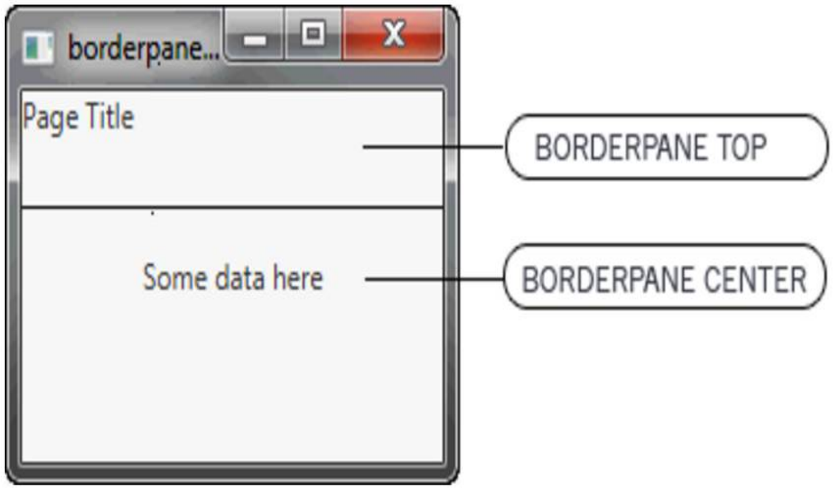
Programmatic vs. Declarative

Programmatic

```
BorderPane border = new BorderPane();

Label top = new Label("Page Title");

border.setTop(top);

Label center = new Label ("Some data here");

border.setCenter(center);
```

Declarative

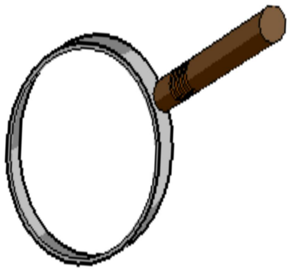
```
<BorderPane>

    <top>

        <Label text="Page Title"/>

    </top>

    <center>

        <Label text="Some data here"/>

    </center>

</BorderPane>
```



FXML structure

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>

<AnchorPane xmlns="http://javafx.com/javafx/8.0.60" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <GridPane hgap="5.0" vgap="5.0">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
          <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
            <Button mnemonicParsing="false" prefHeight="25.0" prefWidth="99.0" text="Login" 
GridPane.rowIndex="2" />
            <Label prefHeight="30.0" prefWidth="98.0" text="Username" />
            <Label prefHeight="40.0" prefWidth="100.0" text="Password" GridPane.rowIndex="1" />
            <TextField GridPane.columnIndex="1" />
            <TextField prefHeight="31.0" prefWidth="100.0" GridPane.columnIndex="1" GridPane.rowIndex="1" />
         </children>
         <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
         </padding>
      </GridPane>
   </children>
</AnchorPane>
```

FXML structure



```
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane>
   <children>
      <GridPane hgap="5.0" vgap="5.0">
        <columnConstraints> </columnConstraints>
        <rowConstraints></rowConstraints>
         <children>
            <Button text="Login" GridPane.rowIndex="2" GridPane.columnIndex=
"0" />
            <Label text="Username"  GridPane.rowIndex="0"
GridPane.columnIndex= "0" />
            <Label text="Password" GridPane.rowIndex="1"GridPane.columnIndex=
"0" />
            <TextField GridPane.columnIndex="1" GridPane.rowIndex="0" />
            <TextField GridPane.columnIndex="1" GridPane.rowIndex="1" />
         </children>
         <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
         </padding>
      </GridPane>
   </children>
</AnchorPane>
```

FXML Loader

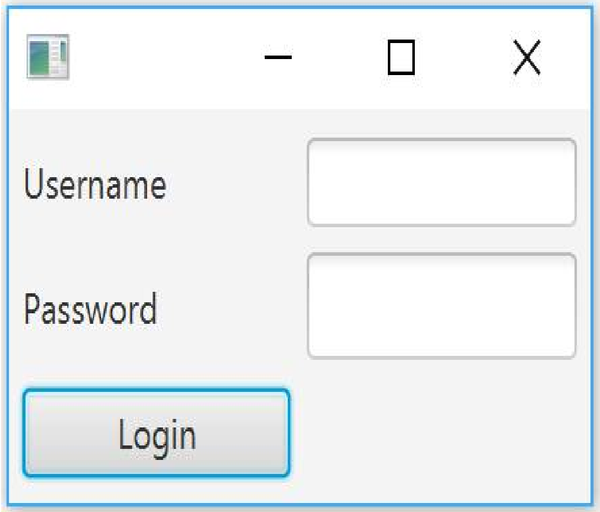
```
public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        try {
            //Load root layout from fxml file.
            FXMLLoader loader=new FXMLLoader();
            // Loads an object hierarchy from an XML document.
            loader.setLocation(Main.class.getResource("View.fxml"));
            //Finds a resource  with a given name  -> URL.
            AnchorPane rootLayout= (AnchorPane) loader.load();
            // Show the scene containing the root layout.
            Scene scene = new Scene(rootLayout);
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Example

View.fxml

```
public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        try {
            //Load root layout from fxml file.
            FXMLLoader loader=new FXMLLoader();

loader.setLocation(Main.class.getResource("View.fxml")
); //URL
            AnchorPane rootLayout= (AnchorPane)
loader.load();

            // Show the scene containing the root
layout.
            Scene scene = new Scene(rootLayout);
            primaryStage.setScene(scene);
            primaryStage.show();

        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane>
   <children>
      <GridPane hgap="5.0" vgap="5.0">
        <columnConstraints> </columnConstraints>
        <rowConstraints></rowConstraints>
         <children>
            <Button text="Login" GridPane.rowIndex="2"
GridPane.columnIndex= "0" />
            <Label text="Username"  GridPane.rowIndex="0"
GridPane.columnIndex= "0" />
            <Label text="Password"
GridPane.rowIndex="1"GridPane.columnIndex= "0" />
            <TextField GridPane.columnIndex="1"
GridPane.rowIndex="0" />
            <TextField GridPane.columnIndex="1"
GridPane.rowIndex="1" />
         </children>
         <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
         </padding>
      </GridPane>
   </children>
</AnchorPane>
```



FXML - Controller

- Define the GUI in the fxml file

- Events are treated in the Controller file. How?

  - Define a file with the name XXXController.java
  - The link to XXX.fxml file is specified as:

    `<AnchorPane fx:controller=" XXXController.java“>`
  - Define handlers in `XXXController.java` to treat the events

FXML – Controller example

```
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane
fx:controller="View2Controller"
>
   <children>
      <GridPane hgap="5.0" vgap="5.0">
        <columnConstraints> </columnConstraints>
        <rowConstraints></rowConstraints>
         <children>
            <Button text="Login" GridPane.rowIndex="2" GridPane.columnIndex= "0" />
            <Label text="Username"  GridPane.rowIndex=“0" GridPane.columnIndex= "0" />
            <Label text="Password" GridPane.rowIndex="1“GridPane.columnIndex= "0" />
            <TextField GridPane.columnIndex=“1" GridPane.rowIndex=“0" />
            <TextField GridPane.columnIndex="1" GridPane.rowIndex="1" />
         </children>
         <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
         </padding>
      </GridPane>
   </children>
</AnchorPane>
```

```
public class View2Controller {

    /**
     * Initializes the controller class. This method is
automatically called
     * after the fxml file has been loaded.
     */
    @FXML
    private void initialize() {

    }
}
```

FXML – Controller- events treatment

```
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane fx:controller="View2Controller">
   <children>
      <GridPane hgap="5.0" vgap="5.0">
        <rowConstraints></rowConstraints>
         <children>
            <Button fx:id="buttonLogin" onAction="#handleLogin" text="Login" GridPane.rowIndex="2"
GridPane.columnIndex= "0" />
            <Label text="Username"  GridPane.rowIndex="0" GridPane.columnIndex= "0" />
            <Label text="Password" GridPane.rowIndex="1"GridPane.columnIndex= "0" />
            <TextField fx:id="textFieldUsername" GridPane.columnIndex="1" GridPane.rowIndex="0" />
            <TextField fx:id="textFieldPasword" GridPane.columnIndex="1" GridPane.rowIndex="1" />
         </children>
         <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
         </padding>
      </GridPane>
   </children>
</AnchorPane>
```

```
public class View2Controller {
    @FXML
    private TextField textFieldUsername;
    @FXML
    private TextField textFieldPasword;

    @FXML
    public void handleLogin() {
        User u=new User(textFieldUsername.getText(),
textFieldPasword.getText());
        //...
    }
}
```

Getting the controller object

```
@Override
public void start(Stage primaryStage) {
    try {
        //Load root layout from fxml file.
        FXMLLoader loader=new FXMLLoader();
        loader.setLocation(Main2.class.getResource("View2.fxml")); //URL
        AnchorPane rootLayout= (AnchorPane) loader.load();

        View2Controller controller=loader.getController();

        // Show the scene containing the root layout.
        Scene scene = new Scene(rootLayout);
        primaryStage.setScene(scene);
        primaryStage.show();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

ObservableValue<T>

- Generic interface **ObservableValue<T>** is used to wrap different values type and to provide a mechanism to observe the values changes through notifications

  public interface ObservableValue<T>extends Observable;

- Methods:

```
T getValue(); //get the wrapped value
```

// adding/removing a ChangeListener (which is notified when the value changes)

```
void addListener(ChangeListener<? super T> listener);

void removeListener(ChangeListener<? super T> listener);
```

Property<T>

- A generic interface that defines the methods common to all properties independent of their type.
- It implements ObservableValue<T>
- Examples of implementations:

public class SimpleStringProperty extends StringPropertyBase;

public class SimpleObjectProperty<T> extends ObjectPropertyBase<T>;

public class SimpleDoubleProperty extends DoublePropertyBase;

Property binding

- JavaFX property binding allows you to synchronize the value of two properties so that whenever one of the properties changes, the value of the other property is updated automatically.
  - Unidirectional binding

    void bind(ObservableValue<? extends T> observable)

  - Bidirectional binding

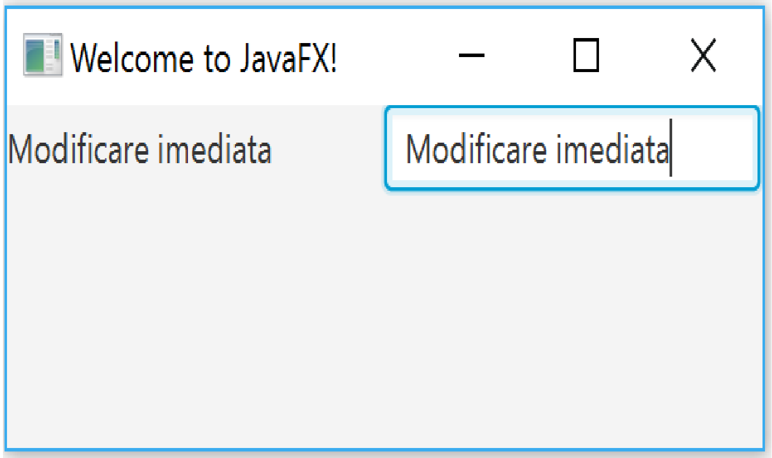
    void bindBidirectional(Property<T> other)

Property - Observable -listener

```
BooleanProperty booleanProperty = new SimpleBooleanProperty(true);
// Add change listener
booleanProperty.addListener(new ChangeListener<Boolean>() {
    @Override
    public void changed(ObservableValue<? extends Boolean> observable,
              Boolean oldValue, Boolean newValue) {
        System.out.println("changed " + oldValue + "->" + newValue);
        //myFunc();
    }
});
Button btn = new Button();
btn.setText("Switch boolean flag");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        booleanProperty.set(!booleanProperty.get()); //switch
        System.out.println("Switch to " + booleanProperty.get());
    }
});

// Bind to another property variable
btn.underlineProperty().bind(booleanProperty); //button text is underlined
according to the booleanProperty value
```

TextField- ChangeListener

```
Label l=new Label("search item ...");
l.setPrefWidth(150);
TextField txt=new TextField();
gr.add(l,0,0);
gr.add(txt,1,0);

txt.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> observable,
String oldValue,      String newValue) { l.setText(newValue);
    }
});
```



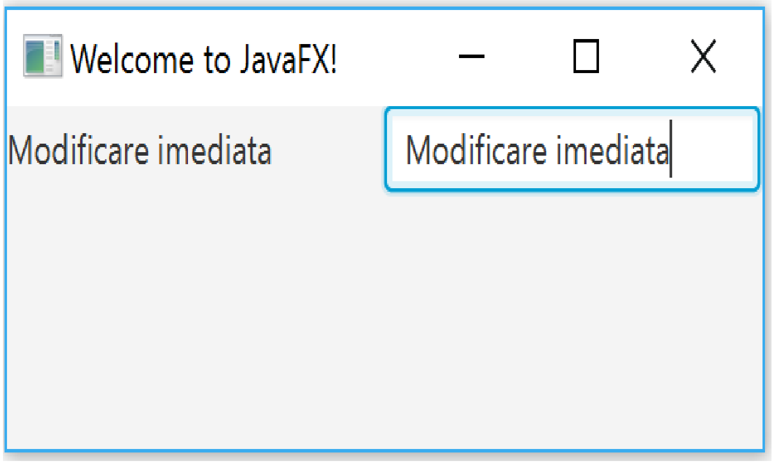
Modificare imediata

Modificare imediata

```
//the same effect using key event handler
txt.setOnKeyPressed(new EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent event) {
        l.setText(txt.getText());
    }
});
```

Or

```
//the same effect using the properties binding
l.textProperty().bindBidirectional(txt.textProperty());
l.setText("third approach");
```



Modificare imediata

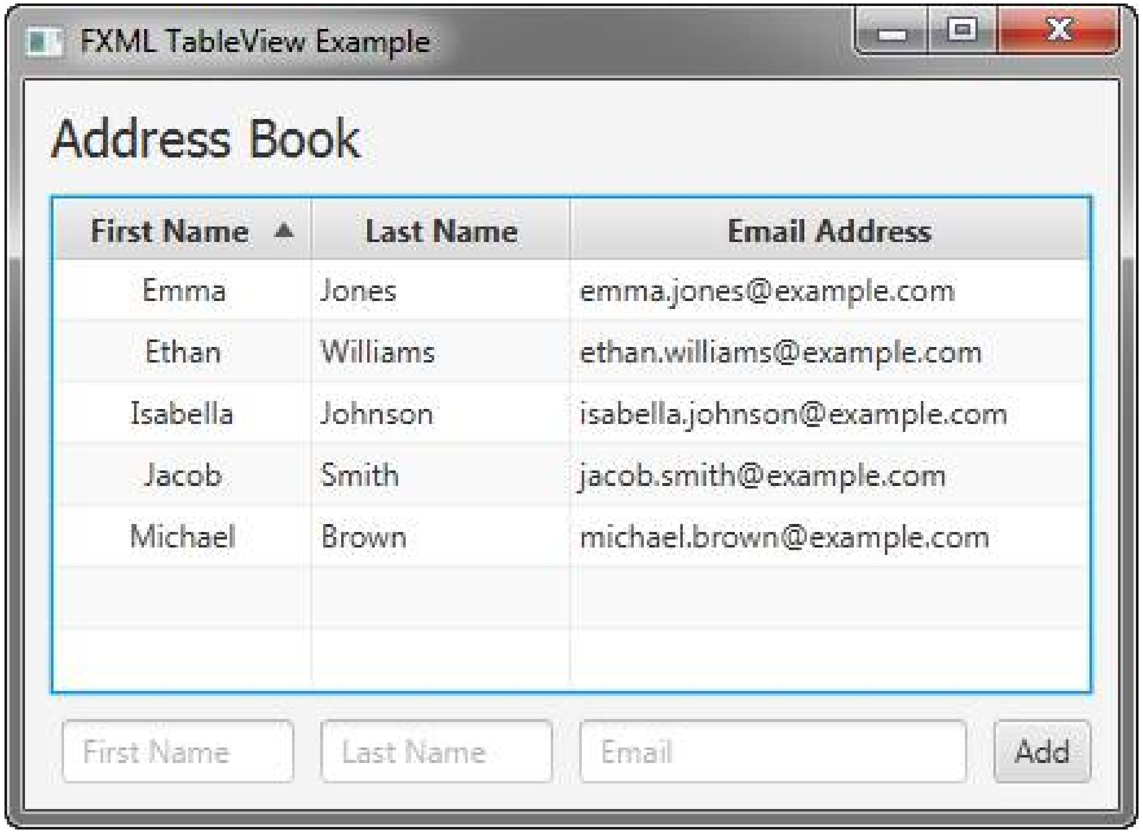
Modificare imediata

Other Examples

- TableView Examples
  - Sorting data
  - Editing data
  - Adding a combo box column
- ListView Example
- Menu Example
- ContextMenu Example
- Opening a new window
- Switching to different screens Example

TableView Example

- This example is based on the Oracle tutorial
- See Ex3-TableView.zip



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

Define the Data Model

- When you create a table in a JavaFX application, it is a best practice to implement a class that defines the data model and provides methods and fields to further work with the table.
- Create a Person class to define the data for the address book.

Controller

- Method initialize
  - Factory methods for each column: see the correspondence between Person fields and table columns
  - Fills the default values

TableView Example Cont.

- See Ex4-TableView.zip
- **Sorting Data in Columns**
  - The TableView class provides built-in capabilities to sort data in columns.
  - Users can alter the order of data by clicking column headers.
  - you can set sorting preferences for each column in your application by applying the setSortType method
  - You can also specify which columns to sort by adding and removing TableColumn instances from the TableView.sortOrder observable list
  - See method changeSorting from TableController class

TableView Example Cont.

- **Editing data in the table**
  - Use the setCellFactory method to reimplement the table cell as a text field with the help of the TextFieldTableCell class.
  - The setOnEditCommit method processes editing and assigns the updated value to the corresponding table cell (requires that users press the Enter key to commit the edit)

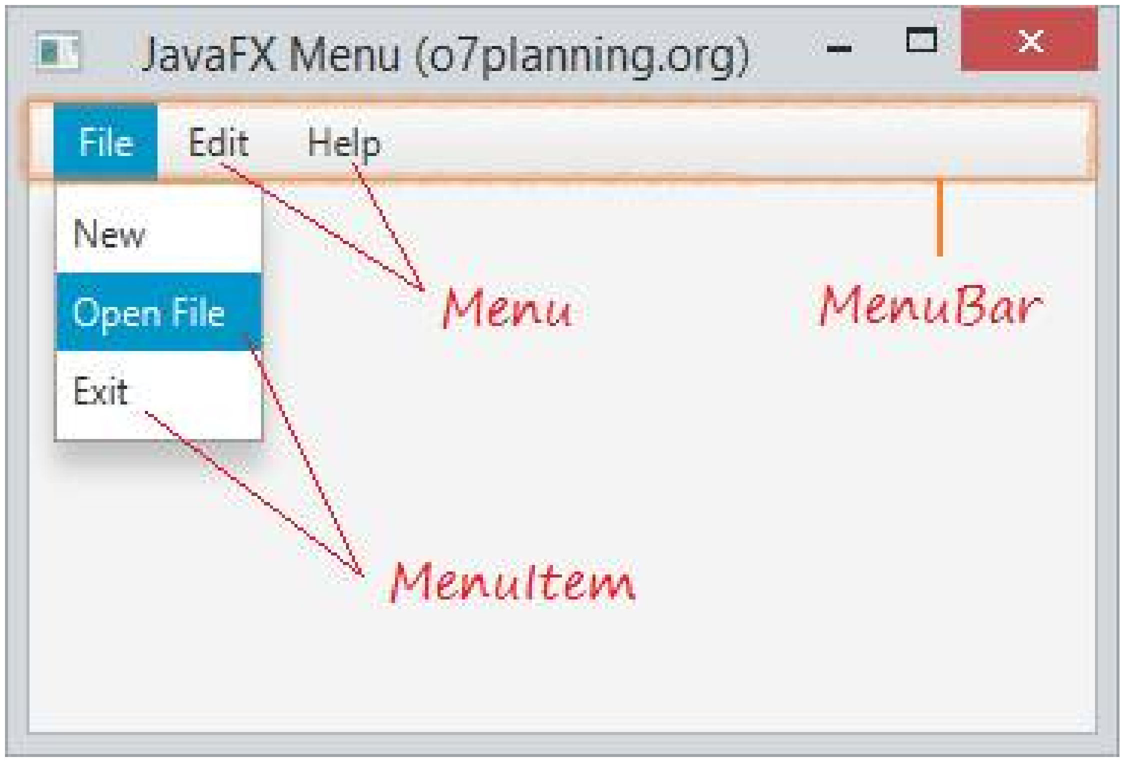
TableView Example Cont.

- **Adding a combo box column**
  - See Gender.java as an enum example
  - Please note that Enum in Java are reference types like class or interface and you can define constructor, methods and variables inside java Enum which makes it more powerful than Enum in C and C++.
  - See the part about combobox in initialize method from TableController class
    - genderColumn.setCellValueFactory
    - genderColumn.setCellFactory
    - genderColumn.setOnEditCommit – for editing

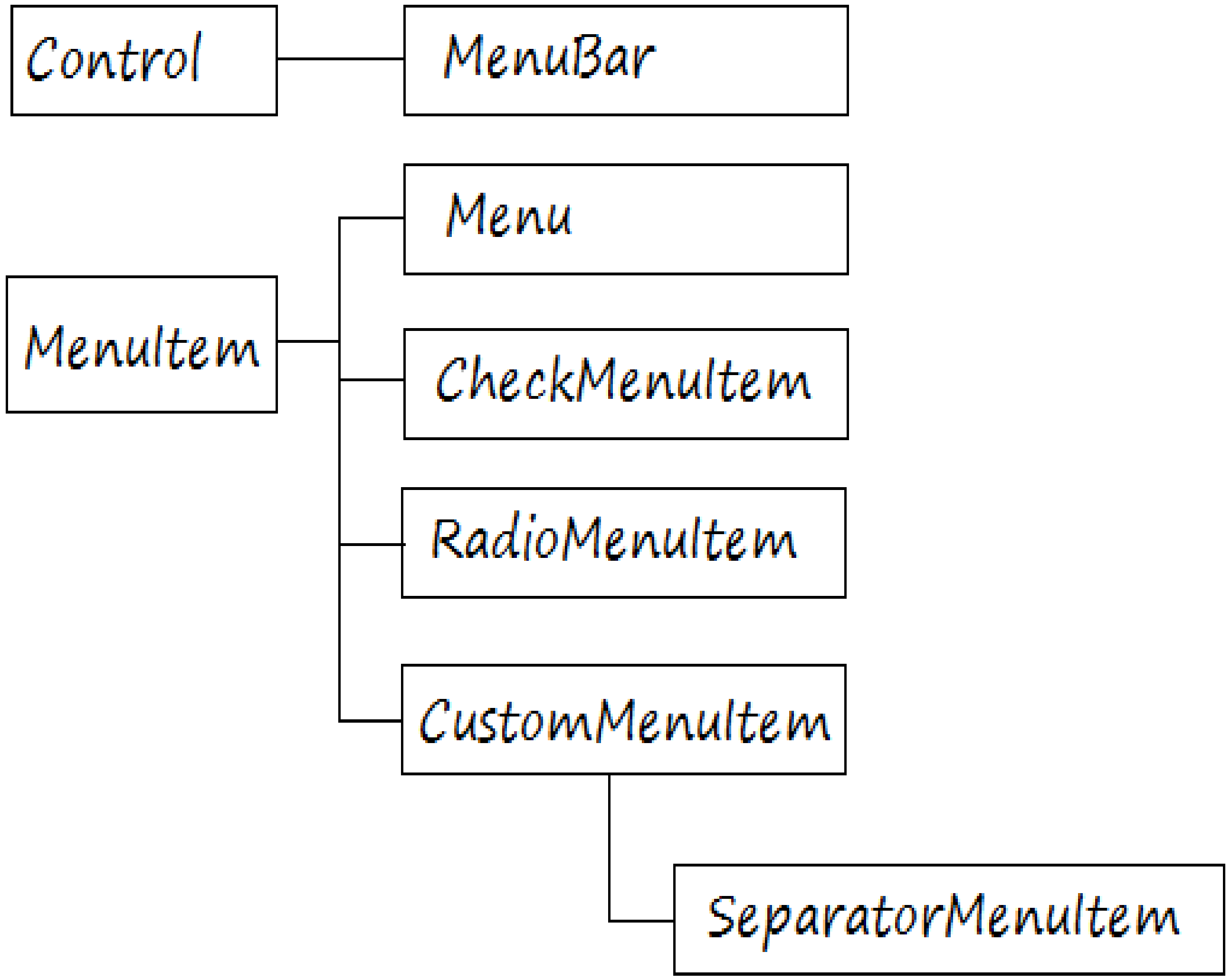
ListView Example

- See Ex5-ListView.zip
- In ListViewController class
  - the list initialization
  - Setting the focus model
  - Setting the selection model
  - Handling the selection

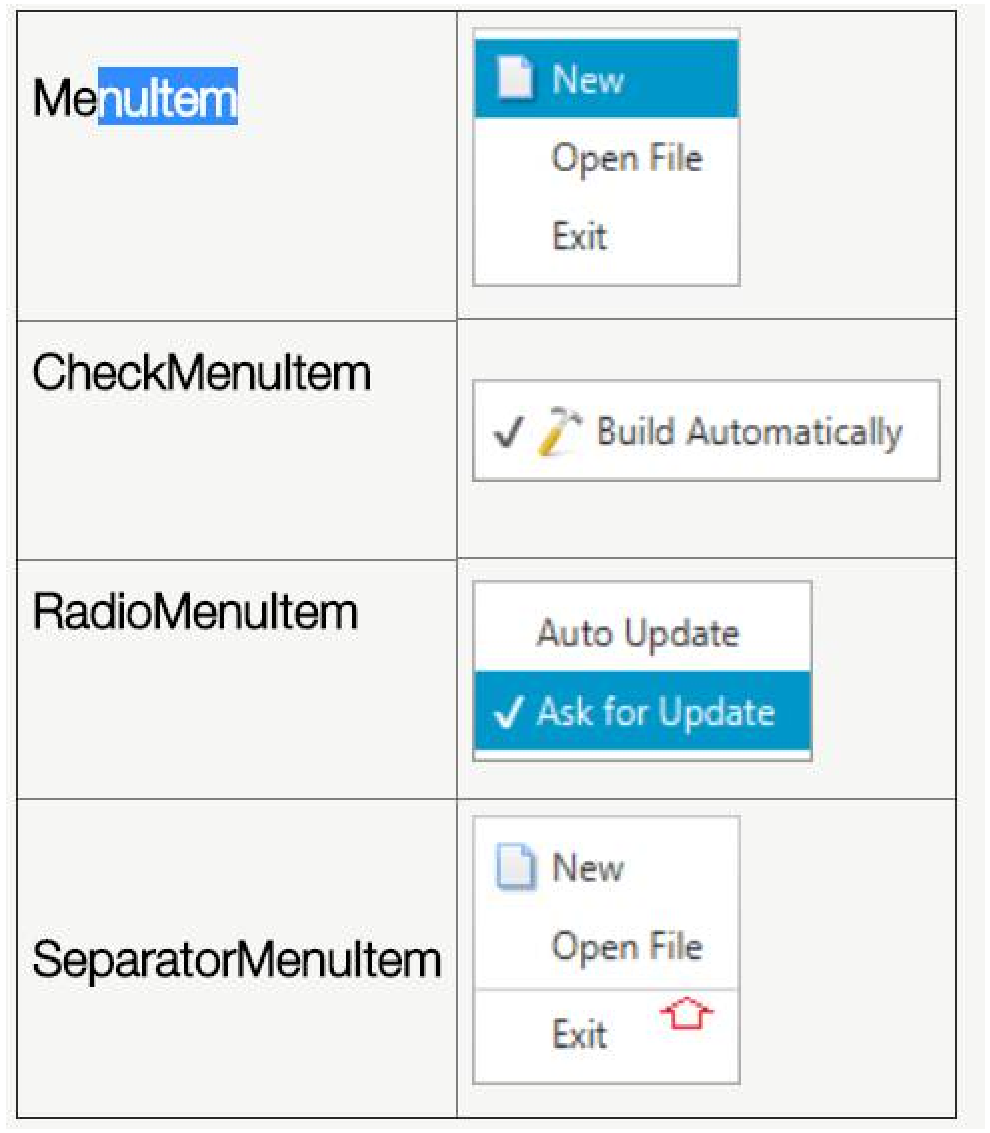
Menu Example



Menu Example



Menu Example



MenuItem	New Open File Exit
CheckMenuItem	✓ Build Automatically
RadioMenuItem	Auto Update ✓ Ask for Update
SeparatorMenuItem	New Open File Exit

Menu Example

- See Ex6-Menu.zip
- Setting the menu structure in SceneBuilder:CheckMenuItem, RadioMenuItem, etc.
- Additional settings for RadioMenuItem in initialize method of MenuController class
- Set accelerator for Exit MenuItem
- Set the handler when click on Exit MenuItem

ContextMenu Example

- **See Ex7-CtxMenu.zip**
- Create a context Menu
- Setting when the context menu is showed

Opening a New Window Example

- When creating a new Stage, you can set up a parent window for it (also called the window owning it), via the stage.initOwner(parentStage)method
- There are three modelities that you can apply to the Stage through the stage.initModality(Modality) method.
  - Modality.NONE:When you open a new window with this modality, the new window will be independent from the parent window. You can interact with the parent window, or close it without affecting the new window.
  - Modality.WINDOW_MODAL:When you open a new window with this modality, it will lock the parent window. You can not interact with the parent window until this window is closed.
  - Modality.APPLICATION_MODAL:When you open a new window with this modality, it will lock any other windows of the application. You can not interact with any other windows until this window is closed.

Opening a New Window Example

- See Ex8-NewWindow.zip
- See how new stages are created in controller
- See how a reference to the primary stage is transmitted to the controller

Switching to different screens

- **See Ex9-SwitchScenes.zip**
- See how a new scene is changed when the button is pressed