Advanced Programming Methods

Lecture 11 - JavaFx(Continuation)

Content

- Event Driven Programming – Event Handling
- A Simple Application without SceneBuilder
- Same Application with FXML (generated by SceneBuilder)
- FXML
- ObservableValue - Property
- A TableView Example

Event Driven Programming

Event: Any user action generates an event:

- pressing or releasing the keyboard,
- moving the mouse,
- pressing or releasing a button mouse,
- opening or closing a window,
- performing a mouse click on a component of the interface,
- entering / leaving the mouse cursor in/out of a component area
- ...

- There are also events that are not generated by the application user.
- An event can be treated by executing a program module.

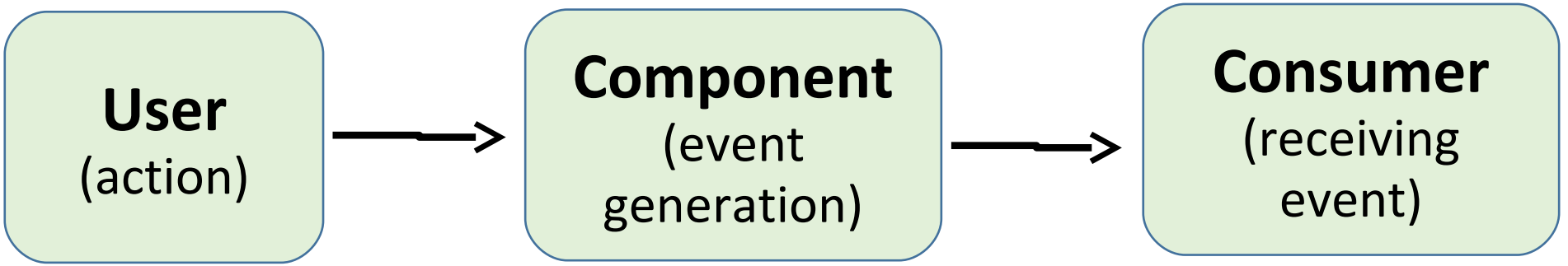
Events Handling

- **Delegation Event Model**

We can distinguish three categories of objects used to handle events:

- **Events Sources** - those objects that generate the events;
- **Events** -which are objects (generated by sources and received by consumers)
- **Events consumers or listeners** - those objects that receive and treat the events.

Events handling

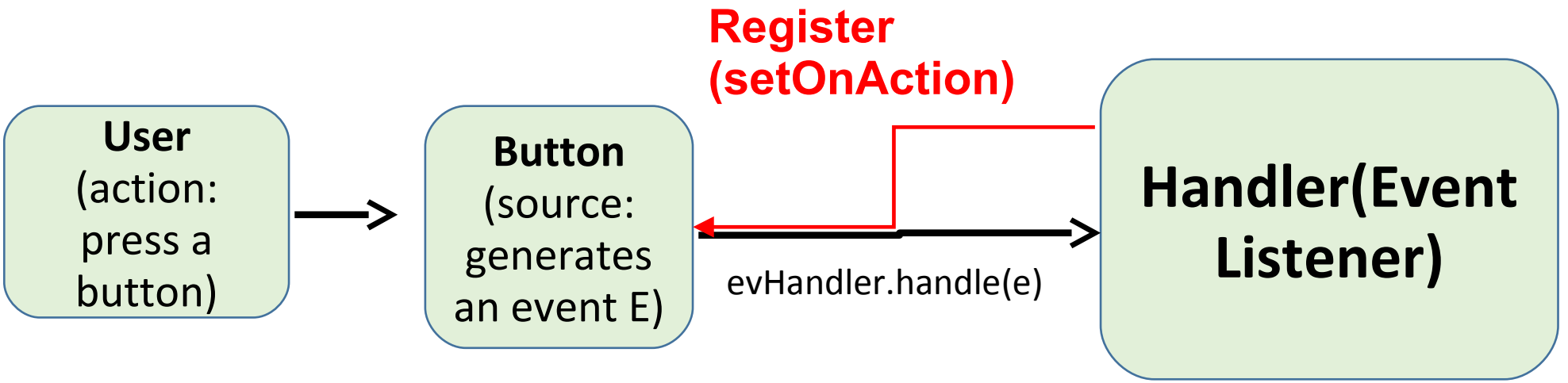


Each consumer must be registered with the event source. This process ensures that the source knows all the consumers to which must submit its generated events.

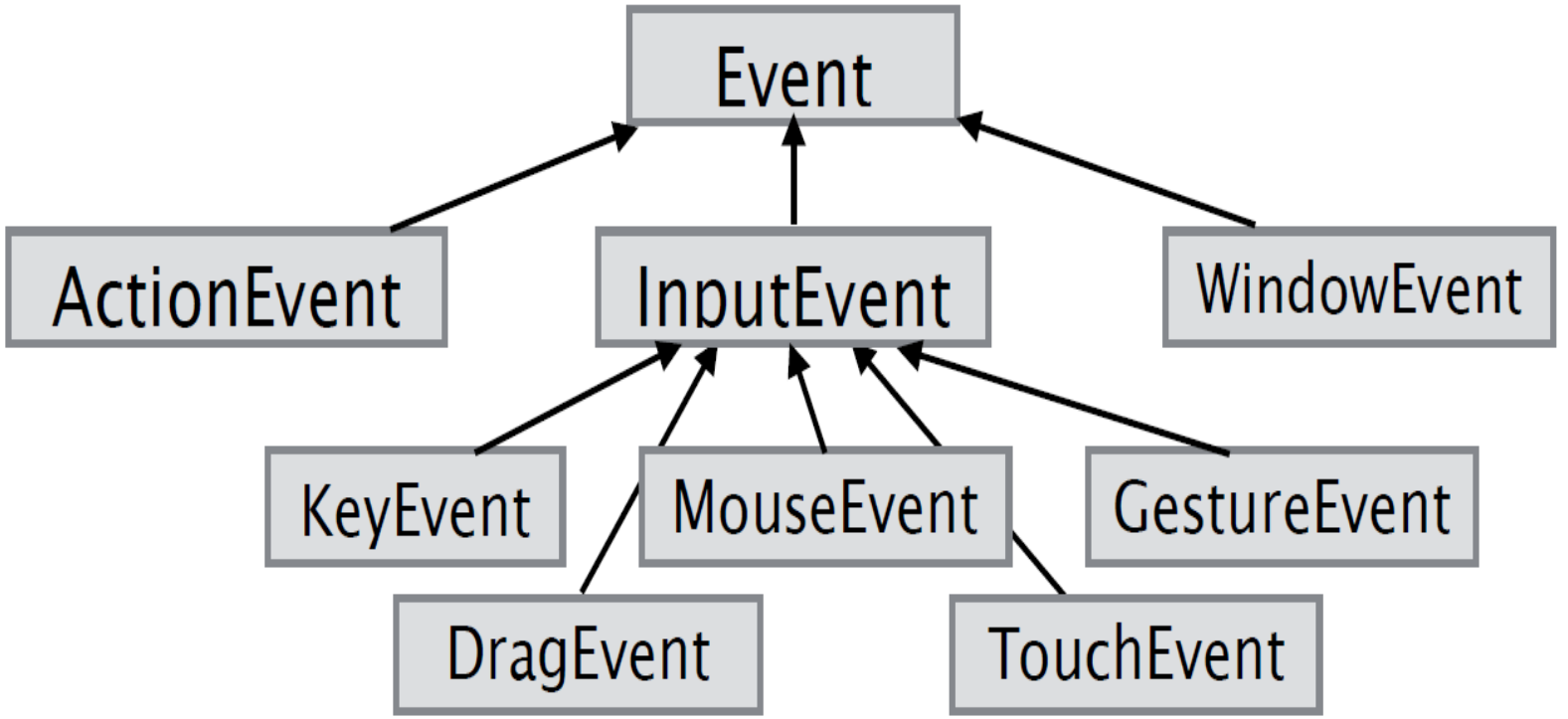
The "delegation" assumes that an event source (an object) transmits all its generated events to those consumers which were recorded at it.

A consumer receives events only from those sources to which it have been registered !!!

Events Handling



Types of Events



Event handler

```
@FunctionalInterface
Interface EventHandler<T extends Event> extends
EventListener{
        void   handle(T event);
}
```

Event handler

Button Events

Only one handler can be associated to the button clicked event!!!

```
Button btn = new Button("Ding!");
btn.setStyle("-fx-font: 42 arial; -fx-base: #f8e7f9;");
// handle the button clicked event
btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {
        System.out.println("Hello World!");
    }
});
```

Or using lambda expressions:

```
btn.setOnAction(e->System.out.println("Hello World!"));
```

See HelloWord.zip

A simple Application

- We follow the Oracle tutorial to create manually (without SceneBuilder) a simple form JavaFx application
- **See Ex1-NoSceneBuilder.zip**

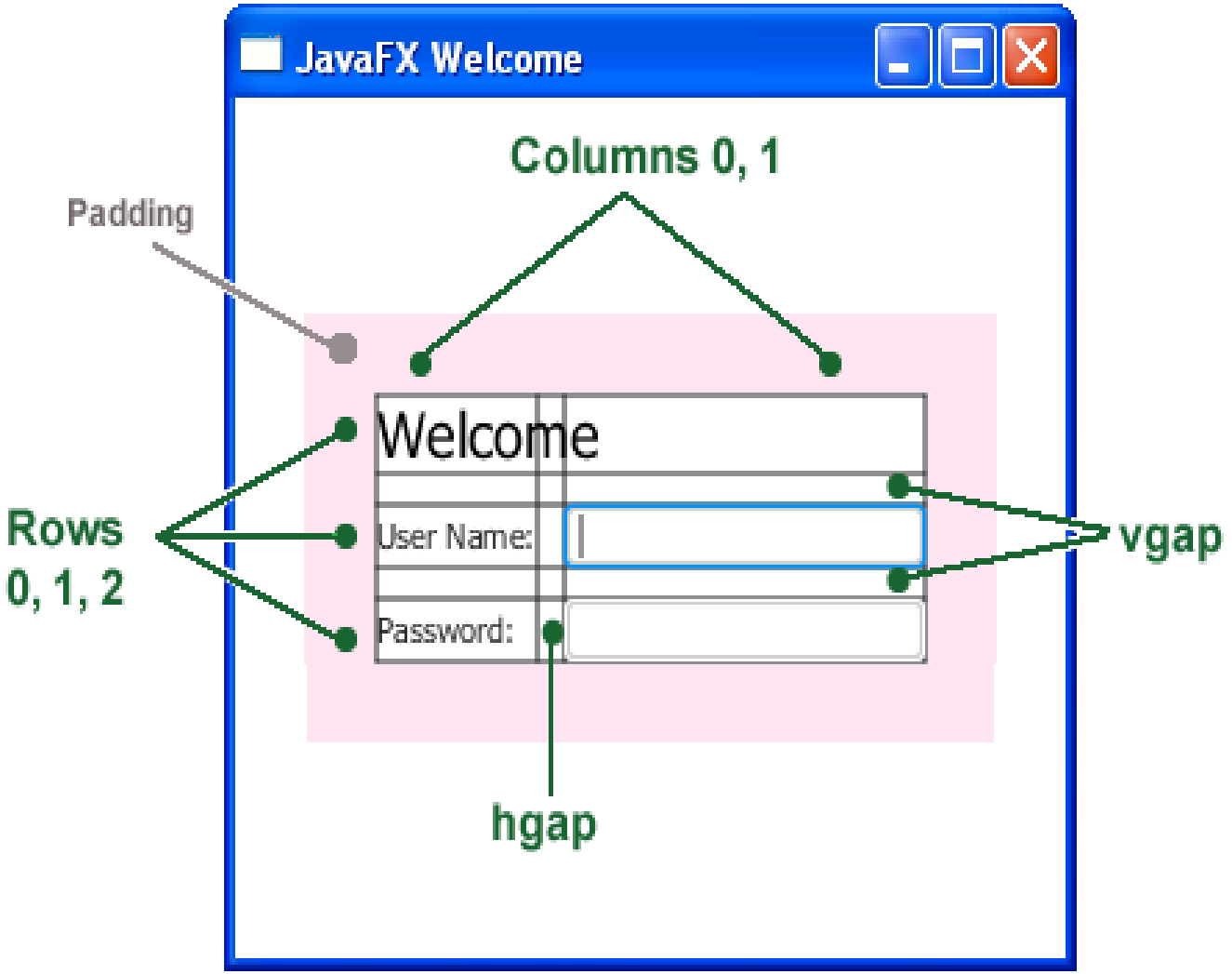
Create a GridPane with Gap and Padding Properties

```
public void start(Stage primaryStage) {

Try {

primaryStage.setTitle("JavaFX Welcome");

GridPane grid = new GridPane();

grid.setAlignment(Pos.CENTER);

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(25, 25, 25, 25));

.......

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

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

primaryStage.setScene(scene);

primaryStage.show();

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

Add Text, Labels, and Text Fields

```
Text scenetitle = new Text("Welcome");
scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
grid.add(scenetitle, 0, 0, 2, 1);

Label userName = new Label("User Name:");
grid.add(userName, 0, 1);

TextField userTextField = new TextField();
grid.add(userTextField, 1, 1);
Label pw = new Label("Password:");
grid.add(pw, 0, 2);
PasswordField pwBox = new PasswordField();
grid.add(pwBox, 1, 2);
```

Add Text, Labels, and Text Fields



Add a Button and Text

```
Button btn = new Button("Sign in");

HBox hbBtn = new HBox(10);

hbBtn.setAlignment(Pos.BOTTOM_RIGHT);

hbBtn.getChildren().add(btn);

grid.add(hbBtn, 1, 4);

final Text actiontarget = new Text();

grid.add(actiontarget, 1, 6);
```

Stylling a button

```
btn.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
```

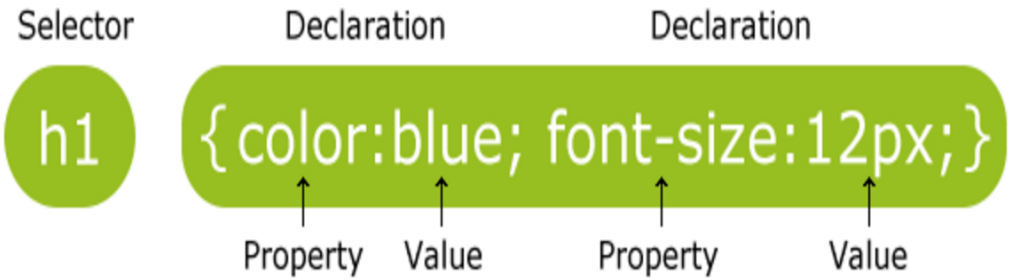
Add a Cascading Style Sheet (CSS)

- Create a .css file and apply the new styles on the previous example
- By switching to CSS over inline styles, we separate the design from the content.
- This approach makes it easier for a designer to have control over the style without having to modify content.
- JavaFx CSS Reference Guide: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

- First: Initialize the stylesheets Variable

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

CSS



http://www.w3schools.com/css/

```
.button {
    -fx-padding: 5 22 5 22;
    -fx-border-color: #e2e2e2;
    -fx-border-width: 2;
    -fx-background-radius: 0;
    -fx-background-color: #1e2e2e2;
    -fx-font-family: "Segoe UI", Helvetica, Arial,
sans-serif;
    -fx-font-size: 11pt;
    -fx-text-fill: black;
    -fx-background-insets: 0 0 0 0, 0, 1, 2;
}

.button:hover {
    -fx-background-color: #3a3a3a;
}
```

```
.background {
    -fx-background-color:
#1d1d1d;
}

.label {
    -fx-font-size: 11pt;
    -fx-font-family: "Segoe UI
Semibold";
    -fx-text-fill: white;
    -fx-opacity: 0.6;
}
```

Add a Background Image

- The background image is applied to the .root style, which means it is applied to the root node of the Scene instance.
- The style definition consists of the name of the property (-fx-background-image) and the value for the property (url(”background.jpg”)).

```
.root {
    -fx-background-image: url("backgr.jpg");
}
```

Style the Labels

- the .label style class will affect all labels in the form

```
.label {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
    -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) ,
0,0,0,1 );
}
```

Style Text

- Remove code that define the inline styles

```
//          scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL,
20));
//          btn.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");
//            actiontarget.setFill(Color.FIREBRICK);
```

- Create an ID for each text node by using the setID() method of the Node class

```
scenetitle.setId("welcome-text");
actiontarget.setId("actiontarget");
```

Style Text

- Add to CSS file

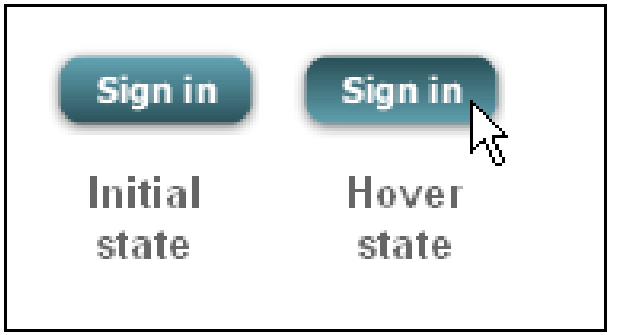
```
#welcome-text {
    -fx-font-size: 32px;
    -fx-font-family: "Arial Black";
    -fx-fill: #818181;
    -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6,
0.0 , 0 , 2 );
}
#actiontarget {
   -fx-fill: FIREBRICK;
   -fx-font-weight: bold;
   -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) ,
0,0,0,1 );
}
```

Style Text

- Text with shadow effects

Style the Button

- Initial state




```
.button {
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5,
0.0 , 0 , 1 );
}
```

- Hover state

```
.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
```

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 code/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

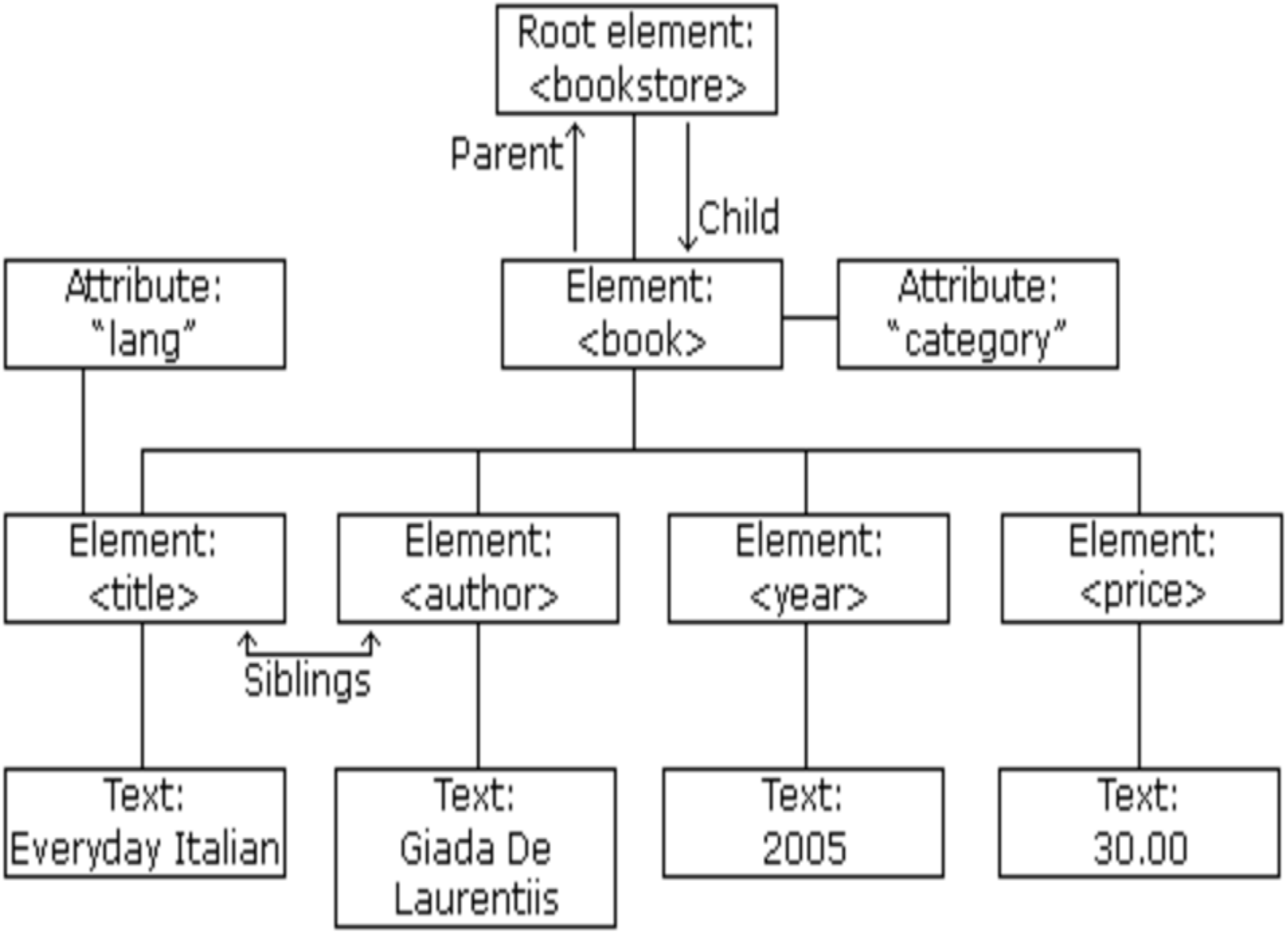
XML stands for EXtensible Markup Language.

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

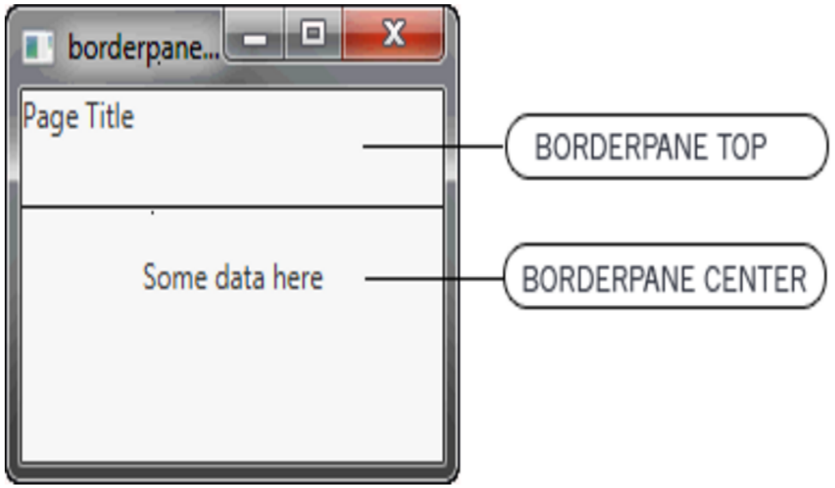
Programmatic vs. Declarative

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 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

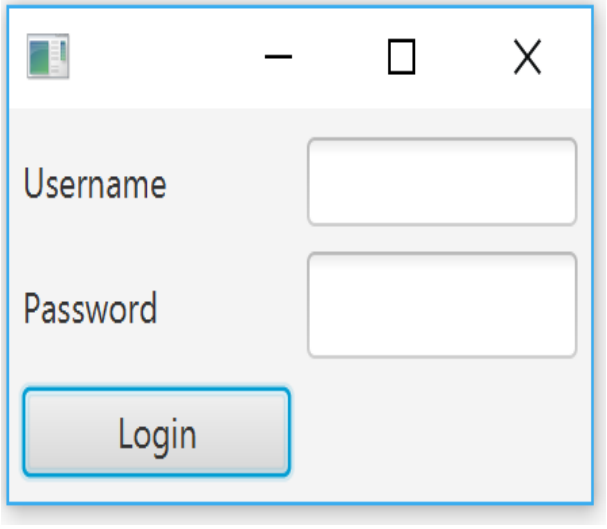
```
public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        try {
            //Load root layout from fxml file.
            FXMLLoader loader=new FXMLLoader();
            // Loads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

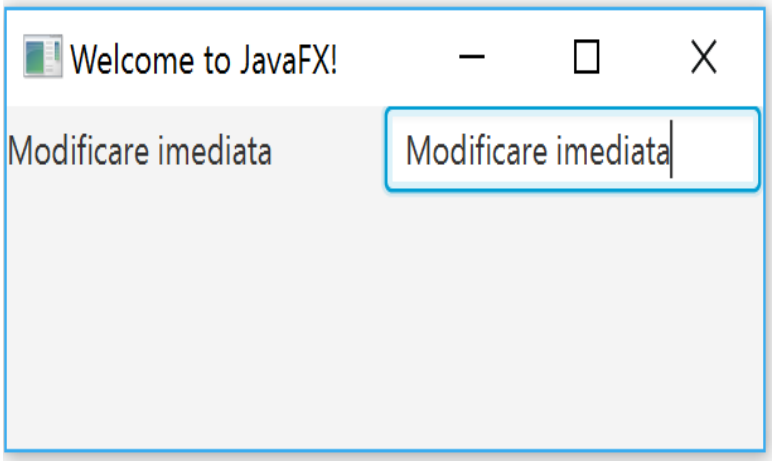
    void bindBidirectional(Property<T> other)

Property - Observable -listener

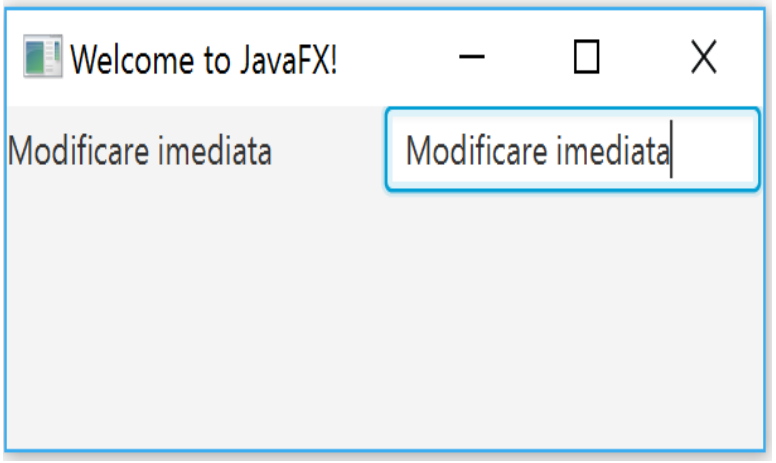
```
BooleanProperty booleanProperty = new SimpleBooleanProperty(true);
// Add change listener
booleanProperty.addListener(new ChangeListener<Boolean>() {
    @Override
    public void changed(ObservableValue<? extends Boolean> observable,
              Boolean oldValue, Boolean newValue) {
        System.out.println("changed " + 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);
    }
});
```

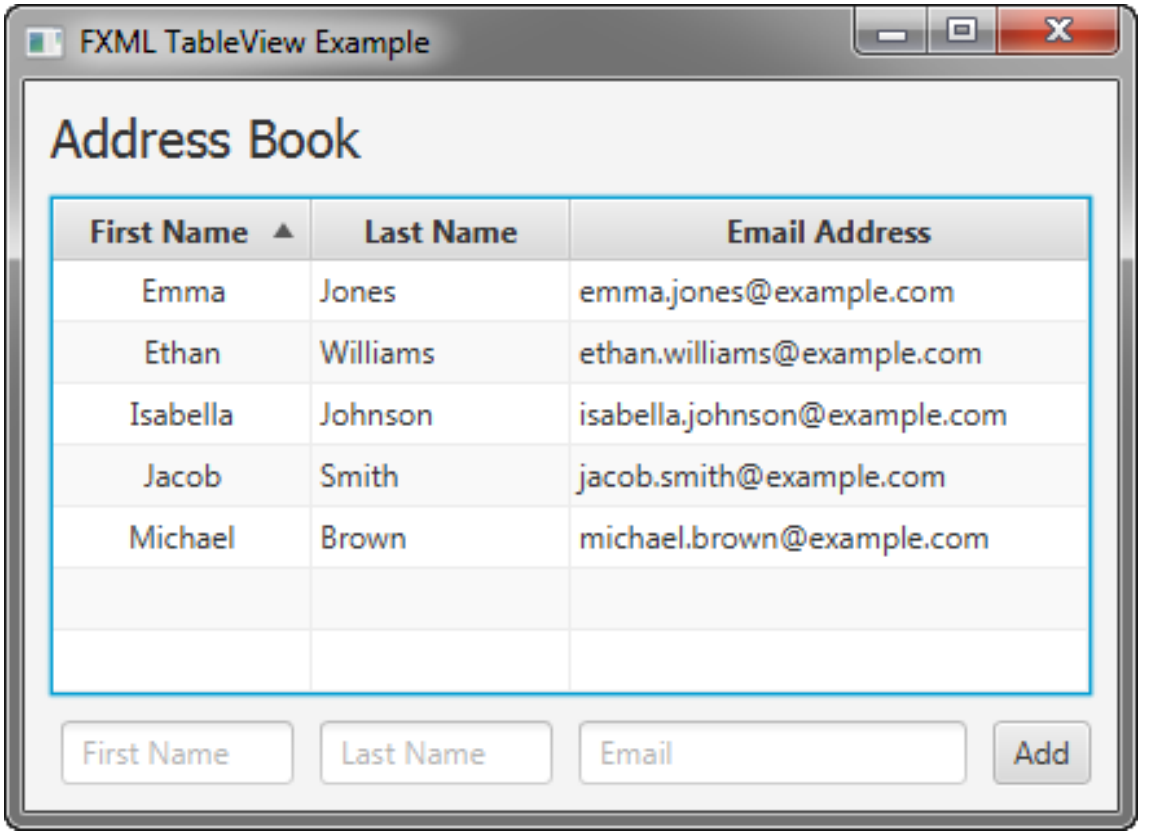


```
//the same effect using key event handler
txt.setOnKeyPressed(new EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent event) {
        l.setText(txt.getText());
    }
});

Or

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



A TableView Example

- We follow the Oracle tutorial to create this example
- **See Ex3-TableView.zip**



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
  - Fills the default values